



Certificate Manager API

Version 1.0
March 6, 2008

**PLEASE REVIEW THE TERMS OF THE FOUNDATION API LICENSE VERSION 1.0
BEFORE READING THIS DOCUMENT.**

You may view the Foundation API License v1.0 at:

http://www.limofoundation.org/images/stories/pdf/limo_foundation_api_license_v1-0.pdf

This is an agreement between you ("You") and LiMo Foundation related to the use of the APIs contained within this document. By accessing, downloading, reading or otherwise using the APIs contained within this document, You accept the terms of the Foundation API License v1.0. If You do not agree to the terms of this agreement, do not download, read and/or otherwise use any of the APIs.

If You are agreeing to be bound by the Foundation API License v1.0 on behalf of your employer, You represent and warrant that you have full legal authority to bind your employer to this agreement. If You do not have the requisite authority, You may not download, read and/or otherwise use any of the APIs on behalf of your employer.

NOTICES

This document and the technical information it contains is subject to change by LiMo Foundation and its members at any time without notice as authorized by LiMo Foundation bylaws and other documents governing LiMo Foundation, including amendments thereto. All statements regarding LiMo's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Any reference in this document to non-LiMo Foundation Web sites do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of this document or any product described in this document and use of those Web sites is at your own risk.

Information concerning non-LiMo products was obtained from the suppliers of those products or a public announcement or source. LiMo cannot confirm the accuracy of that information or the performance, compatibility or any other claims related to non-LiMo products.

All names or other identifying information used in examples in this document are fictitious and any similarity to the names or other identifying information of an actual person or organization is entirely coincidental.

THIS DOCUMENT AND ALL INFORMATION IN THIS DOCUMENT ARE PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OR CONDITION, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OR CONDITION OF ACCURACY OR COMPLETENESS, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT. ALL EXPRESS AND IMPLIED STATUTORY WARRANTIES ARE DISCLAIMED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT.

Contents

1	Certificate Manager API	1
1.1	References	1
1.2	Overview	1
1.3	Industry Standard	5
1.4	External API Documents	5
1.5	Plug-in Extension Point Interface	5
1.6	Other Interfaces	5
1.7	Other Notes	5
2	Data Structure Index	7
2.1	Data Structures	7
3	File Index	9
3.1	File List	9
4	Data Structure Documentation	11
4.1	_CertMgrCertLinkedList Struct Reference	11
4.2	CertMgrAdditionalFields Struct Reference	12
4.3	CertMgrCertDescriptor Struct Reference	13
4.4	CertMgrCertFieldDesc Struct Reference	14
4.5	CertMgrCertId Struct Reference	15
4.6	CertMgrCtx Struct Reference	16
4.7	CertMgrFilBuff Struct Reference	18
4.8	CertMgrKeyDescr Struct Reference	19
4.9	CertMgrMandatoryFields Struct Reference	20
4.10	CertMgrMemBuff Struct Reference	21
4.11	CertMgrMetaDataId Struct Reference	22
4.12	CertMgrPlugInDescr Struct Reference	23
4.13	CertMgrValidityFldData Struct Reference	24

5	File Documentation	25
5.1	CertMgr.h File Reference	25
5.2	CertMgrCertProcessing.h File Reference	52
5.3	CertMgrCryptography.h File Reference	57
5.4	CertMgrDebug.h File Reference	65
5.5	CertMgrInternal.h File Reference	66
5.6	CertMgrPlugin.h File Reference	69
5.7	CertMgrRevocationStatus.h File Reference	71
5.8	CertMgrSignature.h File Reference	73
5.9	CertMgrStore.h File Reference	80

Chapter 1

Certificate Manager API

1.1 References

[1] *Software Architecture Document*, Version 1.1 Revision 1.4, LiMo Foundation, 11 September 2007

1.2 Overview

The Certificate Manager implements the administration and processing of security certificates for a particular system. The main services supplied by the certificate are:

- Certificate Processing (parsing, verification etc).
- Certificate Storage.
- Certificate Meta data administration.
- Certificate Revocation discovery.
- Limited cryptographic services.
- Digital signature processing.

1.2.1 Certificate Manager Architecture

The Certificate Manager services are split into Foundation and Framework sections. The Foundation consists of a set of plug-in modules which implement the services listed above. The Framework is used to implement common services and the routing and control of service requests to the plug-in modules.

The following plug-ins have been defined:

- Cryptography Plug-in
- Certificate Processing Plug-in
- Digital Signature Plug-in
- Certificate Revocation Status Plug-in
- Certificate Store Plug-in

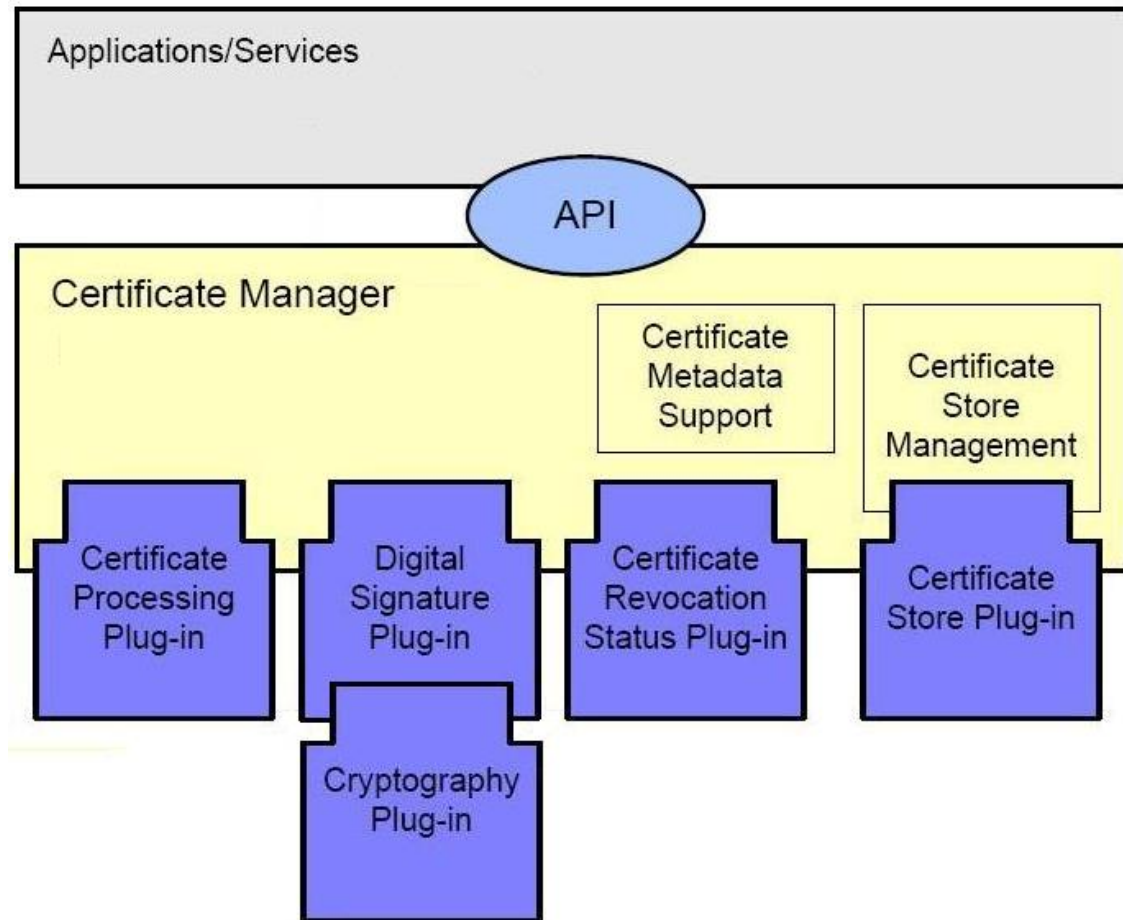


Figure 1.1: Certificate Manager Architecture Diagram

Each plug-in is a separate Linux shared object. The actual shared object that implements a plug-in is specified in the Certificate Manager configuration file. When the Certificate Manager requires access to a particular plug-in, it will search the configuration file for the appropriate entry. Once found, the Certificate Manager will load the shared object and call the required API functions. For the Certificate Processing and Certificate Store plug-ins, multiple instances can be defined for use at the same time. This allows extra functionality to be added easily. For example, there may be an existing Certificate Processing plug-in that understands X.509 certificates. If WTLS certificates are also required, a new shared object (conforming to the Certificate Processing Framework API) can be written. This shared object can be added to the platform installation and the Certificate Manager configuration updated. There will now be two Certificate Processing plug-ins listed in the configuration file: one for X.509 and one for WTLS. The Certificate Manager can use either plug-in when attempting to process a certificate.

1.2.2 Certificate Verification Diagram

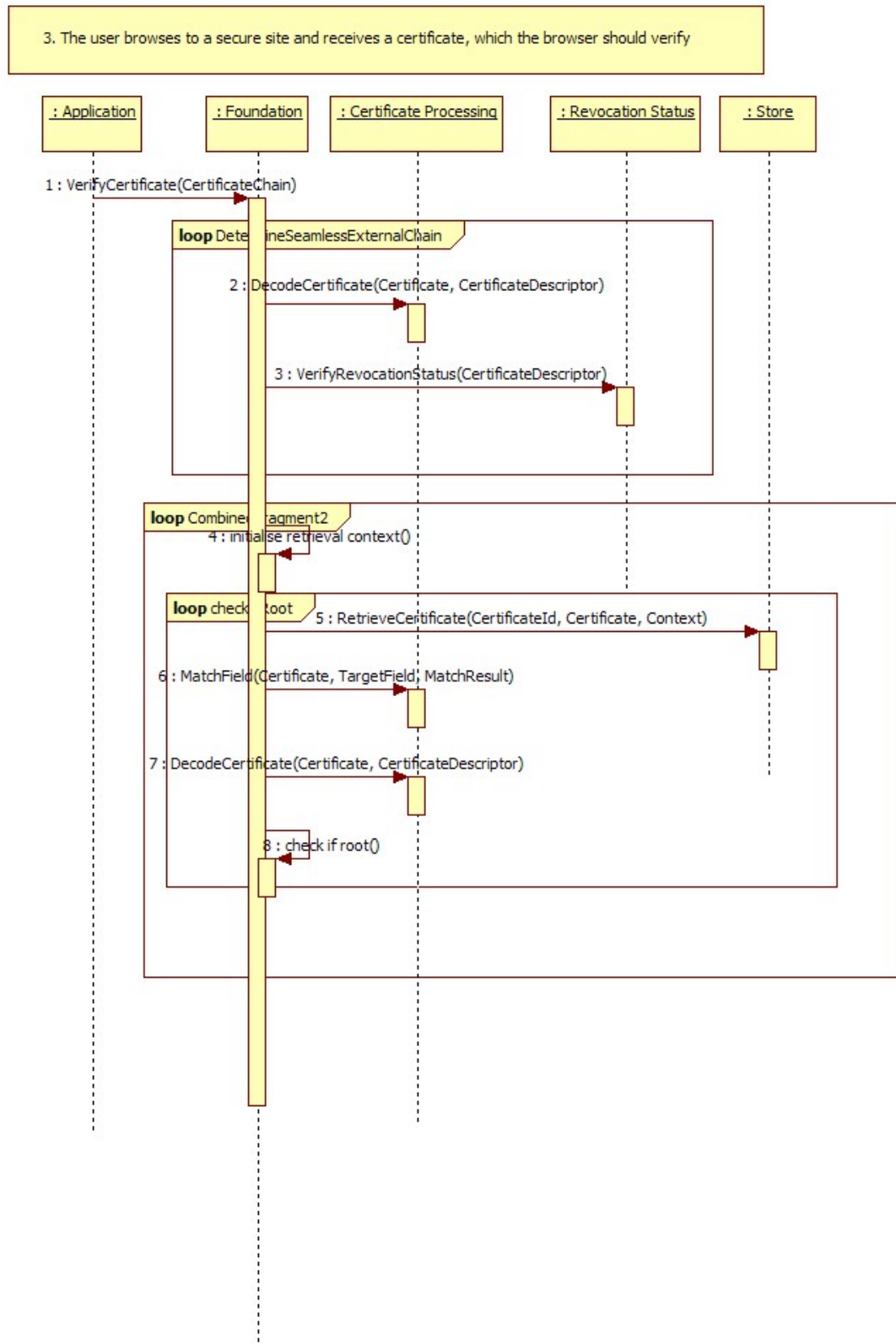


Figure 1.2: Example Sequence Diagram for Certificate Verification

1.3 Industry Standard

None

1.4 External API Documents

None

1.5 Plug-in Extension Point Interface

None

1.6 Other Interfaces

None

1.7 Other Notes

None

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_CertMgrCertLinkedList	11
CertMgrAdditionalFields	12
CertMgrCertDescriptor	13
CertMgrCertFieldDesc	14
CertMgrCertId	15
CertMgrCtx	16
CertMgrFilBuff	18
CertMgrKeyDescr	19
CertMgrMandatoryFields	20
CertMgrMemBuff	21
CertMgrMetaDataId	22
CertMgrPlugInDescr	23
CertMgrValidityFldData	24

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

CertMgr.h (This is the header file for the LiMo Certificate Manager Foundation API)	25
CertMgrCertProcessing.h (This file specifies the certificate-processing plug in component for certificate manager. This plug in is responsible for verifying and parsing of certificates)	52
CertMgrCryptography.h (This file specifies the crypto graphic services supplied by the crypto-graphic plug in component for the certificate manager)	57
CertMgrDebug.h (This is the header file for the LiMo Certificate Manager Foundation API) . .	65
CertMgrInternal.h (This is the header file for the LiMo Certificate Manager Foundation API) . .	66
CertMgrPlugin.h (This is the header file for LiMo Certificate Manager Plug-in framework API)	69
CertMgrRevocationStatus.h (This file specifies the revocation status plug in componenet for certificate manager. This plug in is responsible for handling certificate status request) . .	71
CertMgrSignature.h (This is the component which implements digital signature capability for the certificate manager)	73
CertMgrStore.h (This file specifies the storage services supplied by the certificate store plug-in component for the certificate manager)	80

Chapter 4

Data Structure Documentation

4.1 `_CertMgrCertLinkedList` Struct Reference

```
#include <CertMgr.h>
```

4.1.1 Detailed Description

Linked list of certificates. This is the structure, which represents a member of the linked list of certificates.

Data Fields

- `CertMgrMemBuff * certificate`
- `_CertMgrCertLinkedList * next`

4.1.2 Field Documentation

4.1.2.1 `CertMgrMemBuff* _CertMgrCertLinkedList::certificate`

certificate data in buffer descriptor

4.1.2.2 `struct _CertMgrCertLinkedList* _CertMgrCertLinkedList::next`

end of list indicated by null value in next

The documentation for this struct was generated from the following file:

- `CertMgr.h`

4.2 CertMgrAdditionalFields Struct Reference

```
#include <CertMgr.h>
```

4.2.1 Detailed Description

some additional fields

Data Fields

- [UINT32 numberOfFields](#)
- [CertMgrCertFieldDesc * fields](#)

4.2.2 Field Documentation

4.2.2.1 [UINT32 CertMgrAdditionalFields::numberOfFields](#)

the number of fields in the field array

4.2.2.2 [CertMgrCertFieldDesc* CertMgrAdditionalFields::fields](#)

an array of field descriptors

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.3 CertMgrCertDescriptor Struct Reference

```
#include <CertMgr.h>
```

4.3.1 Detailed Description

parsed certificate content

Data Fields

- `char * type`
- `CertMgrMandatoryFields` **mandatory**
- `CertMgrAdditionalFields` **additional**

4.3.2 Field Documentation

4.3.2.1 `char* CertMgrCertDescriptor::type`

the certificate type identifier

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.4 CertMgrCertFieldDesc Struct Reference

```
#include <CertMgr.h>
```

4.4.1 Detailed Description

a generic structure to define a field in a certificate

Data Fields

- [CertMgrCertField name](#)
- void * [data](#)
- [UINT32 dataSize](#)

4.4.2 Field Documentation

4.4.2.1 [CertMgrCertField CertMgrCertFieldDesc::name](#)

the name of the certificate field

4.4.2.2 void* [CertMgrCertFieldDesc::data](#)

the address of the field data

4.4.2.3 [UINT32 CertMgrCertFieldDesc::dataSize](#)

the size of the field data

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.5 CertMgrCertId Struct Reference

```
#include <CertMgr.h>
```

4.5.1 Detailed Description

The certificate id determined by the API

Data Fields

- [CertMgrStorageId storeId](#)
- [UINT32 certificateNo](#)
- [char * type](#)

4.5.2 Field Documentation

4.5.2.1 [CertMgrStorageId CertMgrCertId::storeId](#)

an identifying string for a certificate store

4.5.2.2 [UINT32 CertMgrCertId::certificateNo](#)

a number identifying a certificate in the store it is stored in

4.5.2.3 [char* CertMgrCertId::type](#)

the certificate type

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.6 CertMgrCtx Struct Reference

```
#include <CertMgr.h>
```

4.6.1 Detailed Description

The context record used to store certificate retrieval context, to be used by successive calls to retrieve certificates

Data Fields

- [BOOL open](#)
- [BOOL firstCall](#)
- [BOOL lastRetrieved](#)
- [BOOL retrieveByField](#)
- [CertMgrCertFieldDesc searchTarget](#)
- [BOOL retrieveByType](#)
- [char * type](#)
- [UINT32 lastRetrievedId](#)
- [CertMgrStorageId storeId](#)

4.6.2 Field Documentation

4.6.2.1 [BOOL CertMgrCtx::open](#)

if TRUE, indicates that the store is open, and certificates can be retrieved

4.6.2.2 [BOOL CertMgrCtx::firstCall](#)

if TRUE, indicates that this is the first call in a sequence of retrieval calls

4.6.2.3 [BOOL CertMgrCtx::lastRetrieved](#)

if TRUE, indicates that the last certificate has been retrieved from store in this retrieval sequence

4.6.2.4 [BOOL CertMgrCtx::retrieveByField](#)

if TRUE, indicates that we are retrieving based on a certificate field (name, subject, etc...)

4.6.2.5 [CertMgrCertFieldDesc CertMgrCtx::searchTarget](#)

if retrieval is by field, this specifies which field we are looking for

4.6.2.6 [BOOL CertMgrCtx::retrieveByType](#)

if TRUE, indicates that we are retrieving based on certificate type

4.6.2.7 char* [CertMgrCtx::type](#)

if retrieving by type, this specifies the certificate type to look for

4.6.2.8 [UINT32 CertMgrCtx::lastRetrievedId](#)

the certificate id of the last retrieved certificate (0 if none retrieved)

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.7 CertMgrFilBuff Struct Reference

```
#include <CertMgr.h>
```

4.7.1 Detailed Description

File data descriptor specifying a file (path-name), an offset from start of file to the first byte of the file record and the size of the file data record in the file

Data Fields

- [char * fileName](#)
- [UINT32 fileOffset](#)
- [UINT32 size](#)

4.7.2 Field Documentation

4.7.2.1 [char* CertMgrFilBuff::fileName](#)

path-name of file

4.7.2.2 [UINT32 CertMgrFilBuff::fileOffset](#)

offset of record from start of file

4.7.2.3 [UINT32 CertMgrFilBuff::size](#)

size of record on file

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.8 CertMgrKeyDescr Struct Reference

```
#include <CertMgr.h>
```

4.8.1 Detailed Description

A key descriptor structure, specifying the key data and size contained inside a memory buffer

Data Fields

- [CertMgrMemBuff * key](#)

4.8.2 Field Documentation

4.8.2.1 [CertMgrMemBuff*](#) [CertMgrKeyDescr::key](#)

key data

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.9 CertMgrMandatoryFields Struct Reference

```
#include <CertMgr.h>
```

4.9.1 Detailed Description

the certificate descriptor in decoded form.

Data Fields

- [CertMgrSigAlgoFldData](#) **sigAlgo**
- [CertMgrIssuerFldData](#) **issuer**
- [CertMgrValidityFldData](#) **validPeriod**
- [CertMgrSubjectFldData](#) **subject**

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.10 CertMgrMemBuff Struct Reference

```
#include <CertMgr.h>
```

4.10.1 Detailed Description

Memory buffer descriptor for a pointer to an array, containing unsigned 8 bit data elements.

Data Fields

- [UINT8 * data](#)
- [UINT32 size](#)
- [UINT32 firstFree](#)

4.10.2 Field Documentation

4.10.2.1 [UINT8* CertMgrMemBuff::data](#)

address of the first buffer location in memory

4.10.2.2 [UINT32 CertMgrMemBuff::size](#)

size of the allocated buffer

4.10.2.3 [UINT32 CertMgrMemBuff::firstFree](#)

starting offset in data buffer available for appending new data

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.11 CertMgrMetaDataId Struct Reference

```
#include <CertMgr.h>
```

4.11.1 Detailed Description

meta data specification

Data Fields

- char * [metaDataType](#)
- char * [ownerApplication](#)

4.11.2 Field Documentation

4.11.2.1 char* [CertMgrMetaDataId::metaDataType](#)

a null terminated string, identifying the type

4.11.2.2 char* [CertMgrMetaDataId::ownerApplication](#)

a null terminated string, identifying the application which created the type

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

4.12 CertMgrPlugInDescr Struct Reference

```
#include <CertMgrSignature.h>
```

4.12.1 Detailed Description

A cryptor plug-in descriptor used to determine which plug-in supports which algorithm.

Data Fields

- [CertMgrCipherAlgo](#) algorithm
- [char * plugInLocation](#)

4.12.2 Field Documentation

4.12.2.1 [CertMgrCipherAlgo](#) [CertMgrPlugInDescr::algorithm](#)

the algorithm supported

4.12.2.2 [char*](#) [CertMgrPlugInDescr::plugInLocation](#)

the path to the plug-in

The documentation for this struct was generated from the following file:

- [CertMgrSignature.h](#)

4.13 CertMgrValidityFldData Struct Reference

```
#include <CertMgr.h>
```

4.13.1 Detailed Description

definition for valid period field data

Data Fields

- [UINT32 firstSecond](#)
- [UINT32 firstHour](#)
- [UINT32 firstDay](#)
- [UINT32 firstMonth](#)
- [UINT32 firstYear](#)
- [UINT32 lastSecond](#)
- [UINT32 lastHour](#)
- [UINT32 lastDay](#)
- [UINT32 lastMonth](#)
- [UINT32 lastYear](#)

The documentation for this struct was generated from the following file:

- [CertMgr.h](#)

Chapter 5

File Documentation

5.1 CertMgr.h File Reference

5.1.1 Detailed Description

This is the header file for the LiMo Certificate Manager Foundation API.

Data Structures

- struct [CertMgrValidityFldData](#)
- struct [CertMgrMemBuff](#)
- struct [CertMgrFilBuff](#)
- struct [CertMgrCertFieldDesc](#)
- struct [CertMgrCtx](#)
- struct [CertMgrMandatoryFields](#)
- struct [CertMgrAdditionalFields](#)
- struct [CertMgrCertDescriptor](#)
- struct [CertMgrCertId](#)
- struct [CertMgrMetaDataId](#)
- struct [_CertMgrCertLinkedList](#)
- struct [CertMgrKeyDescr](#)

Defines

- #define [MAX_ERRCODES_PER_COMPONENT](#) ((INT32)255)
number of error codes reserved for each component
- #define [MAX_RESERVED_ERR_CODE](#) ((INT32)-255)
first error code available the certificate manager sub system
- #define [CERTMGR_START_ERRCODE](#) ((INT32)(MAX_RESERVED_ERR_CODE - (0 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the certificate manager foundation module

- #define **CERTPROCESS_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (1 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the processing framework module
- #define **CERTCRYPTO_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (2 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the crypto graphic framework module
- #define **CERTSIGNATURE_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (3 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the signature framework module
- #define **CERTMETADATA_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (4 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the meta data framework module
- #define **CERTREVOC_STATUS_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (5 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the revocation status framework module
- #define **CERTSTORE_START_ERRCODE** ((INT32)(MAX_RESERVED_ERR_CODE - (6 * MAX_ERRCODES_PER_COMPONENT)))
first error code available to the certificate store framework module
- #define **TRUE** ((int)1)
- #define **FALSE** (!(TRUE))
- #define **ERR_UNKNOWN_RESULT_CODE** ((INT32)(CERTMGR_START_ERRCODE - 0))
function return error code indicating the result code was unknown
- #define **ERR_BROKEN_EXTERN_CHAIN** ((INT32)(CERTMGR_START_ERRCODE - 1))
function return error code indicating the specified certificates could not be arranged into a certificate chain
- #define **ERR_NO_ROOT_CERTIFICATE** ((INT32)(CERTMGR_START_ERRCODE - 2))
function return error code indicating a root certificate did not exist
- #define **ERR_INVALID_SIGNATURE** ((INT32)(CERTMGR_START_ERRCODE - 3))
function return error code indicating an invalid signature
- #define **ERR_INVALID_CERTIFICATE** ((INT32)(CERTMGR_START_ERRCODE - 4))
function return error code indicating an invalid certificate
- #define **ERR_FILE_IO_ERROR** ((INT32)(CERTMGR_START_ERRCODE - 5))
function return error code indicating a file input/output error occurred
- #define **ERR_UNSUPPORTED_HASH_TYPE** ((INT32)(CERTMGR_START_ERRCODE - 6))
function return error code indicating the specified hash type is not supported
- #define **ERR_UNSUPPORTED_KEY_TYPE** ((INT32)(CERTMGR_START_ERRCODE - 7))
function return error code indicating the specified key type is not supported

- #define [ERR_UNSUPPORTED_OPERATION](#) ((INT32)(CERTMGR_START_ERRCODE - 8))
function return error code indicating the specified operation is not supported
- #define [ERR_STORE_IS_FULL](#) ((INT32)(CERTMGR_START_ERRCODE - 9))
function return error code indicating no storage space is available
- #define [ERR_INVALID_CERT_ID](#) ((INT32)(CERTMGR_START_ERRCODE - 10))
function return error code indicating the specified certificate does not exist
- #define [ERR_BUFFER_TOO_SMALL](#) ((INT32)(CERTMGR_START_ERRCODE - 11))
function return error code indicating the specified buffer is too small
- #define [ERR_STORE_IS_EMPTY](#) ((INT32)(CERTMGR_START_ERRCODE - 12))
function return error code indicating the store is empty
- #define [ERR_NO_MORE_CERTIFICATES](#) ((INT32)(CERTMGR_START_ERRCODE - 13))
function return error code indicating there are no more certificates to be retrieved
- #define [ERR_STORE_IS_UNAVAILABLE](#) ((INT32)(CERTMGR_START_ERRCODE - 14))
function return error code indicating the specified store is not available
- #define [ERR_DUPLICATE_CERTIFICATE](#) ((INT32)(CERTMGR_START_ERRCODE - 15))
function return error code indicating a duplicate certificate exists already
- #define [CERTMGR_HASH_MD2_RSA](#) "md2WithRSAEncryption"
a signature based on MD2 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)
- #define [CERTMGR_HASH_MD5_RSA](#) "md5WithRSAEncryption"
a signature based on MD5 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)
- #define [CERTMGR_HASH_SHA1_RSA](#) "sha-1WithRSAEncryption"
a signature based on SHA1 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)
- #define [CERTMGR_CIPHER_RSA](#) "RSA"
cipher algorithm based on RSA
- #define [CERTMGR_CERTFLD_VERSION](#) "version"
a certificate field name specification for the version field
- #define [CERTMGR_CERTFLD_SERIALNO](#) "serial number"
a certificate field name specification for the serial number field
- #define [CERTMGR_CERTFLD_SIG_ALGO](#) "signing algorithm"
a certificate field name specification for the signature algorithm field
- #define [CERTMGR_CERTFLD_ISSUER](#) "issuer"
a certificate field name specification for the issuer field
- #define [CERTMGR_CERTFLD_VALIDITY](#) "valid period"
a certificate field name specification for the end of validity field

- #define CERTMGR_CERTFLD_SUBJECT "subject"
a certificate field name specification for the subject field
- #define CERTMGR_CERTFLD_PUBLIC_KEY "public key"
a certificate field name specification for the public key field
- #define CERTMGR_CERTFLD_ISSUER_UID "issuer UID"
a certificate field name specification for the issuer UID field
- #define CERTMGR_CERTFLD_SUBJECT_UID "subject UID"
a certificate field name specification for the subject UID field
- #define CERTMGR_CERTFLD_EXTENSIONS "extensions"
a certificate field name specification for the extensions field
- #define CERTMGR_CERTFLD_SIGNED_DATA "certificate information"
a certificate field name specification for the certificate information field
- #define CERTMGR_CERTFLD_SIGNATURE "certificate signature"
a certificate field name specification for the certificate signature field
- #define CERTMGR_CERTID_HASH_SIZE 16
the size of the hash identifying each certificate under the control of the certificate manager
- #define CERTMGR_INIT_CONTEXT(context, size) memset((void*)context, 0, size);

Typedefs

- typedef unsigned char UINT8
definition for unsigned 8 bit integer
- typedef char INT8
definition for signed 8 bit integer
- typedef unsigned int UINT16
definition for unsigned 16 bit integer
- typedef int INT16
definition for signed 16 bit integer
- typedef unsigned long int UINT32
definition for unsigned 32 bit integer
- typedef long int INT32
definition for signed 32 bit integer
- typedef int BOOL
- typedef INT32 CertMgrResult

- typedef char * [CertMgrVersionFldData](#)
definition for version field data
- typedef char * [CertMgrSerialNoFldData](#)
definition for serial number field data
- typedef char * [CertMgrSigAlgoFldData](#)
definition for signature algorithm field data
- typedef char * [CertMgrIssuerFldData](#)
definition for issuer field data
- typedef char * [CertMgrSubjectFldData](#)
definition for subject field data
- typedef char * [CertMgrPublicKeyFldData](#)
definition for public key field data
- typedef char * [CertMgrIssuerUIDFldData](#)
definition for issuer UID field data
- typedef char * [CertMgrSubjectUIDFldData](#)
definition for subject UID field data
- typedef char * [CertMgrExtensionsFldData](#)
definition for extensions field data
- typedef char * [CertMgrSigType](#)
definition for the signature hash/key combination
- typedef char * [CertMgrCipherAlgo](#)
definition for cipher algorithms
- typedef char * [CertMgrStorageId](#)
definition for the supported certificate store types
- typedef char * [CertMgrCertStatus](#)
definition for the certificate status
- typedef char * [CertMgrCertField](#)
definition for a certificate field identifier
- typedef [_CertMgrCertLinkedList](#) [CertMgrCertLinkedList](#)

Functions

- [CertMgrResult](#) [CertMgrErrorCodeToMessage](#) ([CertMgrResult](#) code, char **resultMessage)
maps the specified error code to an error message string.

- [CertMgrResult CertMgrGetCertificateType](#) ([CertMgrMemBuff](#) *certificate, char *certificateType)
Determines the type of the certificate.
- [CertMgrResult CertMgrVerifyCertificate](#) ([CertMgrCertLinkedList](#) *certificateList)
Verifies a certificate chain, consisting of one or more certificates.
- [CertMgrResult CertMgrVerifyMBufSigWithCert](#) (char *certificateType, [CertMgrMemBuff](#) *certificate, [CertMgrMemBuff](#) *message, [CertMgrMemBuff](#) *signature)
Uses a security certificate to verify that the specified signature matches a specified message stored in memory. The certificate determines the signature algorithm and the public key to decode the signature.
- [CertMgrResult CertMgrVerifyFBufSigWithCert](#) (char *certificateType, [CertMgrMemBuff](#) *certificate, [CertMgrFilBuff](#) *msgInFile, [CertMgrMemBuff](#) *signature)
Uses a security certificate to verify that a specified signature matches the specified message stored in a file record. The certificate determines the signature algorithm and the public key to decode the signature.
- [CertMgrResult CertMgrVerifyMemBufferSignature](#) ([CertMgrSigType](#) algorithm, [CertMgrKeyDescr](#) *keyDescriptor, [CertMgrMemBuff](#) *message, [CertMgrMemBuff](#) *signature)
Uses a hash type specification and a key descriptor to verify that a specified signature matches the specified message stored in a memory buffer.
- [CertMgrResult CertMgrVerifyFileBufferSignature](#) ([CertMgrSigType](#) algorithm, [CertMgrKeyDescr](#) *keyDescriptor, [CertMgrFilBuff](#) *msgInFile, [CertMgrMemBuff](#) *signature)
Uses a hash type specification and a key descriptor to verify that a specified signature matches the specified message stored in a file record.
- [CertMgrResult CertMgrGenerateMemBuffSignature](#) ([CertMgrSigType](#) algorithm, void *keyId, [CertMgrMemBuff](#) *message, [CertMgrMemBuff](#) *signature)
this function will generate a signature over a specified memory buffer.
- [CertMgrResult CertMgrGenerateFilBuffSignature](#) ([CertMgrSigType](#) algorithm, void *keyId, [CertMgrFilBuff](#) *msgInFile, [CertMgrMemBuff](#) *signature)
this function will generate a signature over a specified file record.
- [CertMgrResult CertMgrAddMetaDataToCert](#) ([CertMgrCertId](#) *certificateId, [CertMgrMetaDataId](#) *metaDataId, [CertMgrMemBuff](#) *metaData)
saves the specified meta data block in the meta data buffer and links the data to a certificate with the specified id.
- [CertMgrResult CertMgrRemoveMetaDataFromCert](#) ([CertMgrCertId](#) *certificateId, [CertMgrMetaDataId](#) *metaDataId)
removes specified meta data from the meta data buffer and unlinks the specified certificate from the removed meta data.
- [CertMgrResult CertMgrPurgeMetaDataFromCert](#) ([CertMgrCertId](#) *certificateId)
removes all meta associated with the specified certificate.
- [CertMgrResult CertMgrRetrieveMetaDataForCert](#) ([CertMgrCertId](#) *certificateId, [CertMgrMetaDataId](#) *metaDataId, [CertMgrMemBuff](#) *metaData)
retrieves the specified meta data type from the meta data buffer.

- **CertMgrResult CertMgrExtractCertificateData** (**CertMgrMemBuff** *certificate, **CertMgrCertDescriptor** *certificateDescriptor)
Assemble a certificate record from raw certificate data.
- **CertMgrResult CertMgrReleaseCertificateData** (**CertMgrCertDescriptor** *certificateDescriptor)
Releases certificate field records allocated in a previous call to CertMgrExtractCertificateData.
- **CertMgrResult CertMgrEnableCertificate** (**CertMgrCertId** *certificateId)
This function will change the state of the specified certificate to enabled.
- **CertMgrResult CertMgrDisableCertificate** (**CertMgrCertId** *certificateId)
This function will change the state of the specified certificate to disabled.
- **CertMgrResult CertMgrRetrieveCertStatus** (**CertMgrCertId** *certificateId, **CertMgrCertStatus** *status)
This function is used to retrieve the certificate status (whether it is enabled or disabled).
- **CertMgrResult CertMgrAddCertificateToStore** (**CertMgrMemBuff** *certificate, **CertMgrCertId** *certificateId, **CertMgrStorageId** storeId)
This function is used to store the certificate memory buffer in the specified store.
- **CertMgrResult CertMgrRemoveCertificateFromStore** (**CertMgrCertId** *certificateId)
This function is used to remove the specified certificate from the specified store.
- **CertMgrResult CertMgrRetrieveCertFromStore** (**CertMgrCtx** *context, **CertMgrMemBuff** *certificate, **CertMgrCertId** *certificateId)
*This function is used to retrieve one or more certificates from the certificate store.
In the first call the context must be properly initialized.*

5.1.2 Define Documentation

5.1.2.1 **#define CERTCRYPTO_START_ERRCODE** ((**INT32**)(MAX_RESERVED_ERR_CODE - (2 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the crypto graphic framework module

5.1.2.2 **#define CERTMETADATA_START_ERRCODE** ((**INT32**)(MAX_RESERVED_ERR_CODE - (4 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the meta data framework module

5.1.2.3 **#define CERTMGR_CERTFLD_EXTENSIONS** "extensions"

a certificate field name specification for the extensions field

5.1.2.4 **#define CERTMGR_CERTFLD_ISSUER** "issuer"

a certificate field name specification for the issuer field

5.1.2.5 #define CERTMGR_CERTFLD_ISSUER_UID "issuer UID"

a certificate field name specification for the issuer UID field

5.1.2.6 #define CERTMGR_CERTFLD_PUBLIC_KEY "public key"

a certificate field name specification for the public key field

5.1.2.7 #define CERTMGR_CERTFLD_SERIALNO "serial number"

a certificate field name specification for the serial number field

5.1.2.8 #define CERTMGR_CERTFLD_SIG_ALGO "signing algorithm"

a certificate field name specification for the signature algorithm field

5.1.2.9 #define CERTMGR_CERTFLD_SIGNATURE "certificate signature"

a certificate field name specification for the certificate signature field

5.1.2.10 #define CERTMGR_CERTFLD_SIGNED_DATA "certificate information"

a certificate field name specification for the certificate information field

5.1.2.11 #define CERTMGR_CERTFLD_SUBJECT "subject"

a certificate field name specification for the subject field

5.1.2.12 #define CERTMGR_CERTFLD_SUBJECT_UID "subject UID"

a certificate field name specification for the subject UID field

5.1.2.13 #define CERTMGR_CERTFLD_VALIDITY "valid period"

a certificate field name specification for the end of validity field

5.1.2.14 #define CERTMGR_CERTFLD_VERSION "version"

a certificate field name specification for the version field

5.1.2.15 #define CERTMGR_CERTID_HASH_SIZE 16

the size of the hash identifying each certificate under the control of the certificate manager

5.1.2.16 #define CERTMGR_CIPHER_RSA "RSA"

cipher algorithm based on RSA

5.1.2.17 #define CERTMGR_HASH_MD2_RSA "md2WithRSAEncryption"

a signature based on MD2 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)

5.1.2.18 #define CERTMGR_HASH_MD5_RSA "md5WithRSAEncryption"

a signature based on MD5 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)

5.1.2.19 #define CERTMGR_HASH_SHA1_RSA "sha-1WithRSAEncryption"

a signature based on SHA1 encrypted with RSA - naming is based on ASN.1 TAG names (see RFC3279)

5.1.2.20 #define CERTMGR_INIT_CONTEXT(context, size) memset((void*)context, 0, size);

macro to initialize a context record to 0

5.1.2.21 #define CERTMGR_START_ERRCODE ((INT32)(MAX_RESERVED_ERR_CODE - (0 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the certificate manager foundation module

5.1.2.22 #define CERTPROCESS_START_ERRCODE ((INT32)(MAX_RESERVED_ERR_CODE - (1 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the processing framework module

5.1.2.23 #define CERTREVOC_STATUS_START_ERRCODE ((INT32)(MAX_RESERVED_ERR_CODE - (5 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the revocation status framework module

5.1.2.24 #define CERTSIGNATURE_START_ERRCODE ((INT32)(MAX_RESERVED_ERR_CODE - (3 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the signature framework module

5.1.2.25 #define CERTSTORE_START_ERRCODE ((INT32)(MAX_RESERVED_ERR_CODE - (6 * MAX_ERRCODES_PER_COMPONENT)))

first error code available to the certificate store framework module

5.1.2.26 `#define ERR_BROKEN_EXTERN_CHAIN ( (INT32)(CERTMGR_START_ERRCODE - 1) )`

function return error code indicating the specified certificates could not be arranged into a certificate chain

5.1.2.27 `#define ERR_BUFFER_TOO_SMALL ( (INT32)(CERTMGR_START_ERRCODE - 11) )`

function return error code indicating the specified buffer is too small

5.1.2.28 `#define ERR_DUPLICATE_CERTIFICATE ( (INT32)(CERTMGR_START_ERRCODE - 15) )`

function return error code indicating a duplicate certificate exists already

5.1.2.29 `#define ERR_FILE_IO_ERROR ( (INT32)(CERTMGR_START_ERRCODE - 5) )`

function return error code indicating a file input/output error occurred

5.1.2.30 `#define ERR_INVALID_CERT_ID ( (INT32)(CERTMGR_START_ERRCODE - 10) )`

function return error code indicating the specified certificate does not exist

5.1.2.31 `#define ERR_INVALID_CERTIFICATE ( (INT32)(CERTMGR_START_ERRCODE - 4) )`

function return error code indicating an invalid certificate

5.1.2.32 `#define ERR_INVALID_SIGNATURE ( (INT32)(CERTMGR_START_ERRCODE - 3) )`

function return error code indicating an invalid signature

5.1.2.33 `#define ERR_NO_MORE_CERTIFICATES ( (INT32)(CERTMGR_START_ERRCODE - 13) )`

function return error code indicating there are no more certificates to be retrieved

5.1.2.34 `#define ERR_NO_ROOT_CERTIFICATE ( (INT32)(CERTMGR_START_ERRCODE - 2) )`

function return error code indicating a root certificate did not exist

5.1.2.35 `#define ERR_STORE_IS_EMPTY ( (INT32)(CERTMGR_START_ERRCODE - 12) )`

function return error code indicating the store is empty

5.1.2.36 `#define ERR_STORE_IS_FULL ( (INT32)(CERTMGR_START_ERRCODE - 9) )`

function return error code indicating no storage space is available

5.1.2.37 `#define ERR_STORE_IS_UNAVAILABLE ( (INT32)(CERTMGR_START_ERRCODE - 14) )`

function return error code indicating the specified store is not available

5.1.2.38 `#define ERR_UNKNOWN_RESULT_CODE ( (INT32)(CERTMGR_START_ERRCODE - 0) )`

function return error code indicating the result code was unknown

5.1.2.39 `#define ERR_UNSUPPORTED_HASH_TYPE ( (INT32)(CERTMGR_START_ERRCODE - 6) )`

function return error code indicating the specified hash type is not supported

5.1.2.40 `#define ERR_UNSUPPORTED_KEY_TYPE ( (INT32)(CERTMGR_START_ERRCODE - 7) )`

function return error code indicating the specified key type is not supported

5.1.2.41 `#define ERR_UNSUPPORTED_OPERATION ( (INT32)(CERTMGR_START_ERRCODE - 8) )`

function return error code indicating the specified operation is not supported

5.1.2.42 `#define MAX_ERRCODES_PER_COMPONENT ((INT32)255)`

number of error codes reserved for each component

5.1.2.43 `#define MAX_RESERVED_ERR_CODE ((INT32)-255)`

first error code available the certificate manager sub system

5.1.2.44 `#define TRUE ((int)1)`

if neither TRUE nor FALSE have been defined, we define both

5.1.3 Typedef Documentation

5.1.3.1 `typedef int BOOL`

generic boolean type to be used as true false indicator.

5.1.3.2 typedef char* [CertMgrCertField](#)

definition for a certificate field identifier

5.1.3.3 typedef struct [_CertMgrCertLinkedList](#) [CertMgrCertLinkedList](#)

Linked list of certificates. This is the structure, which represents a member of the linked list of certificates.

5.1.3.4 typedef char* [CertMgrCertStatus](#)

definition for the certificate status

5.1.3.5 typedef char* [CertMgrCipherAlgo](#)

definition for cipher algorithms

5.1.3.6 typedef char* [CertMgrExtensionsFldData](#)

definition for extensions field data

5.1.3.7 typedef char* [CertMgrIssuerFldData](#)

definition for issuer field data

5.1.3.8 typedef char* [CertMgrIssuerUIDFldData](#)

definition for issuer UID field data

5.1.3.9 typedef char* [CertMgrPublicKeyFldData](#)

definition for public key field data

5.1.3.10 typedef [INT32](#) [CertMgrResult](#)

function return type.

5.1.3.11 typedef char* [CertMgrSerialNoFldData](#)

definition for serial number field data

5.1.3.12 typedef char* [CertMgrSigAlgoFldData](#)

definition for signature algorithm field data

5.1.3.13 typedef char* CertMgrSigType

definition for the signature hash/key combination

5.1.3.14 typedef char* CertMgrStorageId

definition for the supported certificate store types

5.1.3.15 typedef char* CertMgrSubjectFldData

definition for subject field data

5.1.3.16 typedef char* CertMgrSubjectUIDFldData

definition for subject UID field data

5.1.3.17 typedef char* CertMgrVersionFldData

definition for version field data

5.1.3.18 typedef int INT16

definition for signed 16 bit integer

5.1.3.19 typedef long int INT32

definition for signed 32 bit integer

5.1.3.20 typedef char INT8

definition for signed 8 bit integer

5.1.3.21 typedef unsigned int UINT16

definition for unsigned 16 bit integer

5.1.3.22 typedef unsigned long int UINT32

definition for unsigned 32 bit integer

5.1.3.23 typedef unsigned char UINT8

definition for unsigned 8 bit integer

5.1.4 Function Documentation

5.1.4.1 **CertMgrResult** CertMgrAddCertificateToStore (**CertMgrMemBuff** * *certificate*, **CertMgrCertId** * *certificateId*, **CertMgrStorageId** *storeId*)

This function is used to store the certificate memory buffer in the specified store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificate input - specifies the certificate (memory buffer) to be stored.

certificateId output - the certificate identifier.

storeId input - the store where the certificate is to be stored.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully added
- ERR_STORE_IS_UNAVAILABLE - the specified store is not available
- ERR_STORE_IS_FULL - the specified store is full
- ERR_DUPLICATE_CERTIFICATE - the submitted certificate is already in store

Prospective Clients:

External Apps.

5.1.4.2 **CertMgrResult** CertMgrAddMetaDataToCert (**CertMgrCertId** * *certificateId*, **CertMgrMetaDataId** * *metaDataId*, **CertMgrMemBuff** * *metaData*)

saves the specified meta data block in the meta data buffer and links the data to a certificate with the specified id.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: meta data is application centric. The meta data id is generated by the application and is guaranteed to be unique over all applications using the meta data facility
- Note: certificate ids are generated by the certificate store and uniquely identify a certificate stored by the certificate manager.

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

metaDataId input - a unique identifier for a meta data type.

metaData input - a data descriptor, specifying a block of data in memory, which represents the meta data to be stored.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the meta data has been added successfully.
- ERR_STORE_IS_FULL - no storage space was available to store the meta data

Prospective Clients:

External Apps.

5.1.4.3 CertMgrResult CertMgrDisableCertificate (CertMgrCertId * certificateId)

This function will change the state of the specified certificate to disabled.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate is disabled.
- ERR_ERR_INVALID_CERT_ID - unknown or invalid certificate ID.

Prospective Clients:

External Apps.

5.1.4.4 CertMgrResult CertMgrEnableCertificate (CertMgrCertId * certificateId)

This function will change the state of the specified certificate to enabled.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate is enabled.
- ERR_ERR_INVALID_CERT_ID - unknown or invalid certificate ID.

Prospective Clients:

External Apps.

5.1.4.5 **CertMgrResult** CertMgrErrorCodeToMessage (**CertMgrResult** code, char **resultMessage)

maps the specified error code to an error message string.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: if no message is associated with a particular result code, the function will return an empty string

Parameters:

code input - a result code, for which the user needs the associated message string.

resultMessage output - a null terminated message string associated with the specified error code.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - message was found.
- ERR_UNKNOWN_RESULT_CODE - the specified code is unknown.

Prospective Clients:

External Apps.

Example of how this function would be called:

```
...  
CertMgrResult result;
```

```

char          *resultMessage;

result = CertMgrErrorCodeToMessage (ERR_UNSUPPORTED_OPERATION, &resultMessage);
if(result == RESULT_CODE_SUCCESS)
    --- resultMessage now contains "unsupported operation"
...

```

5.1.4.6 **CertMgrResult** CertMgrExtractCertificateData (**CertMgrMemBuff** * *certificate*, **CertMgrCertDescriptor** * *certificateDescriptor*)

Assemble a certificate record from raw certificate data.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: this function must always be used together with CertMgrReleaseCertificateData
- Note: the certificate field records are allocated in this function, to release the allocated memory use CertMgrReleaseCertificateData.

Warning:

n/a

Parameters:

certificate input - a certificate of any type in raw data.

certificateDescriptor output - the extracted certificate fields from the raw certificate.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified raw certificate is decoded successfully.
- ERR_INVALID_CERTIFICATE - unknown or incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.1.4.7 **CertMgrResult** CertMgrGenerateFilBuffSignature (**CertMgrSigType** *algorithm*, void * *keyId*, **CertMgrFilBuff** * *msgInFile*, **CertMgrMemBuff** * *signature*)

this function will generate a signature over a specified file record.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

- algorithm* input - a hash algorithm specifier for the signature hash generation.
- keyId* input - a key identifier for secure key data access. NOTE: The format of keyId is dependent on the specific platform-dependent implementation of secure key handling used by the Cryptography plug-in.
- msgInFile* input - a message specification record specifying the message stored in a file.
- signature* output - a buffer receiving the generated signature.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature has been successfully generated.
- ERR_UNSUPPORTED_OPERATION - the signature generation has not been implemented.
- ERR_UNSUPPORTED_HASH_TYPE - the specified hash algorithm is not supported.
- ERR_UNSUPPORTED_KEY_TYPE - the specified key type is not supported.
- ERR_FILE_IO_ERROR - problem with file access.

Prospective Clients:

External Apps.

Example of how this function would be called:

```
...
CertMgrResult result;
certMgrMemBuff signature;
...
... do some initialization (signature buffer allocation)

result = CertMgrGenerateMemBuffSignature(CERTMGR_HASH_MD5, key, messageInFile, &signature);

if(result == OPERATION_SUCCESS)
...
```

5.1.4.8 **CertMgrResult** CertMgrGenerateMemBuffSignature (**CertMgrSigType** *algorithm*, void * *keyId*, **CertMgrMemBuff** * *message*, **CertMgrMemBuff** * *signature*)

this function will generate a signature over a specified memory buffer.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

- algorithm* input - a hash algorithm specifier for the signature hash generation.
- keyId* input - a key identifier for secure key data access. NOTE: The format of keyId is dependent on the specific platform-dependent implementation of secure key handling used by the Cryptography plug-in.

message input - a message specification record representing the message for which the signature is to be generated.

signature output - a buffer receiving the generated signature.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature has been successfully generated.
- ERR_UNSUPPORTED_OPERATION - the signature generation has not been implemented.
- ERR_UNSUPPORTED_HASH_TYPE - the specified hash algorithm is not supported.
- ERR_UNSUPPORTED_KEY_TYPE - the specified key type is not supported.

Prospective Clients:

External Apps.

Example of how this function would be called:

```
...
CertMgrResult result;
certMgrMemBuff signature;
...
... do some initialization (signature buffer allocation)

result = CertMgrGenerateMemBuffSignature(CERTMGR_HASH_MD5, key, message, &signature);

if(result == OPERATION_SUCCESS)
...
```

5.1.4.9 CertMgrResult CertMgrGetCertificateType (CertMgrMemBuff * certificate, char * certificateType)

Determines the type of the certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes: n/a**Warning:**

n/a

Parameters:

certificate input - a raw certificate.

certificateType output - the certificate type.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the type was successfully determined.
- ERR_UNKNOWN_CERTIFICATE - the certificate type is unknown.
- ERR_CERTIFICATE_FORMAT_ERROR - the certificate was incorrectly formatted.

Prospective Clients:

External Apps.

5.1.4.10 CertMgrResult CertMgrPurgeMetaDataFromCert (CertMgrCertId * certificateId)

removes all meta associated with the specified certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: meta data is application centric. The meta data id is generated by the application and is guaranteed to be unique over all applications using the meta data facility
- Note: certificate ids are generated by the certificate store and uniquely identify a certificate stored by the certificate manager.

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the meta data has been successfully removed.
- ERR_INVALID_CERT_ID - the specified certificate did not exist in the meta data store.

Prospective Clients:

External Apps.

5.1.4.11 CertMgrResult CertMgrReleaseCertificateData (CertMgrCertDescriptor * certificateDescriptor)

Releases certificate field records allocated in a previous call to CertMgrExtractCertificateData.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateDescriptor input - a certificate descriptor.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the memory allocated for the certificate field records has been successfully released.

Prospective Clients:

External Apps.

5.1.4.12 CertMgrResult CertMgrRemoveCertificateFromStore (CertMgrCertId * certificateId)

This function is used to remove the specified certificate from the specified store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - specifies the certificate to be removed.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate was successfully removed
- Otherwise return the appropriate error

Prospective Clients:

External Apps.

5.1.4.13 CertMgrResult CertMgrRemoveMetaDataFromCert (CertMgrCertId * certificateId, CertMgrMetaDataId * metaDataId)

removes specified meta data from the meta data buffer and unlinks the specified certificate from the removed meta data.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: meta data is application centric. The meta data id is generated by the application and is guaranteed to be unique over all applications using the meta data facility

- Note: certificate ids are generated by the certificate store and uniquely identify a certificate stored by the certificate manager.

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

metaDataId input - a unique identifier a meta data block in the meta data store.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the meta data has been successfully removed.
- ERR_INVALID_CERT_ID - the specified certificate did not exist in the meta data store.

Prospective Clients:

External Apps.

5.1.4.14 **CertMgrResult** CertMgrRetrieveCertFromStore (**CertMgrCtx** * *context*, **CertMgrMemBuff** * *certificate*, **CertMgrCertId** * *certificateId*)

This function is used to retrieve one or more certificates from the certificate store.

In the first call the context must be properly initialized.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: The context is used to keep track of the retrieval operation over successive calls
- The context must be initialized to zero during the first call of this function.
- Use the CERTMGR_INIT_CONTEXT macro to initialize the context.
- Additionally, if the retrieval is to be by field, the field descriptor and the retrieveByField flag must be set appropriately.

Parameters:

context input - The retrieval context.

certificate output - specifies the certificate memory buffer retrieved.

certificateId output - specifies the certificate ID retrieved.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate status successfully retrieved
- ERR_NO_MORE_CERTIFICATES - all certificates have been retrieved
- Otherwise return one appropriate error

Prospective Clients:

External Apps.

5.1.4.15 **CertMgrResult** CertMgrRetrieveCertStatus (**CertMgrCertId** * *certificateId*, **CertMgrCertStatus** * *status*)

This function is used to retrieve the certificate status (whether it is enabled or disabled).

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - a certificate identifier unique over all certificate stores

status output - status of current certificate (enabled or disabled).

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate status successfully retrieved
- Otherwise return the appropriate error

Prospective Clients:

External Apps.

5.1.4.16 **CertMgrResult** CertMgrRetrieveMetaDataForCert (**CertMgrCertId** * *certificateId*, **CertMgrMetaDataId** * *metaDataId*, **CertMgrMemBuff** * *metaData*)

retrieves the specified meta data type from the meta data buffer.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: meta data is application centric. The meta data id is generated by the application and is guaranteed to be unique over all applications using the meta data facility
- Note: certificate ids are generated by the certificate store and uniquely identify a certificate stored by the certificate manager.

Parameters:

certificateId input - a certificate identifier unique over all certificate stores.

metaDataId input - a unique identifier a meta data block in the meta data store.

metaData output - a buffer descriptor for the buffer receiving the retrieved meta data.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the meta data has been retrieved successfully.
- ERR_INVALID_CERT_ID - the specified certificate has no entry in the meta data store
- ERR_BUFFER_TOO_SMALL - the meta data did not fit into the supplied buffer

Prospective Clients:

External Apps.

5.1.4.17 CertMgrResult CertMgrVerifyCertificate (CertMgrCertLinkedList * *certificateList*)

Verifies a certificate chain, consisting of one or more certificates.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: the certificate chain consists of certificates supplied by the caller and are not part of the certificate manager's store
 - to validate the chain, the function must be able to trace the chain from the last in the external chain to an existing trusted root certificate in the certificate manager's store
 - the linked list of certificates does not need to represent the actual verification scheme (certificate->issuer->certificat...).
- It is sufficient that the set of certificates in the linked list can be rearranged into a proper verification chain

Warning:

n/a

Parameters:

certificateList input - linked list of certificates, holding all certificates necessary to form a contiguous part of a verification chain (see notes above).

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - an unbroken verification chain of certificates to a trusted root certificate exists.
- ERR_BROKEN_EXTERN_CHAIN - the members of the linked list could not be re-arranged so that all were contained in the verification chain.
- ERR_NO_ROOT_CERTIFICATE - it was impossible to verify the certificate chain to a trusted root certificate.

Prospective Clients:

External Apps.

5.1.4.18 **CertMgrResult** CertMgrVerifyFBufSigWithCert (char * *certificateType*, **CertMgrMemBuff** * *certificate*, **CertMgrFilBuff** * *msgInFile*, **CertMgrMemBuff** * *signature*)

Uses a security certificate to verify that a specified signature matches the specified message stored in a file record. The certificate determines the signature algorithm and the public key to decode the signature.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateType input - certificate type.

certificate input - a security certificate of any type, used to verify the message signature.

msgInFile input - the file record descriptor representing the message belonging to the signature.

signature input - the signature specification which applies to the message.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature is a match for the specified message.
- ERR_INVALID_SIGNATURE - specified signature does not match the specified message.
- ERR_INVALID_CERTIFICATE - unknown or incorrectly formatted certificate.
- ERR_FILE_IO_ERROR - a file access error occurred.

Prospective Clients:

External Apps.

5.1.4.19 **CertMgrResult** CertMgrVerifyFileBufferSignature (**CertMgrSigType** *algorithm*, **CertMgrKeyDescr** * *keyDescriptor*, **CertMgrFilBuff** * *msgInFile*, **CertMgrMemBuff** * *signature*)

Uses a hash type specification and a key descriptor to verify that a specified signature matches the specified message stored in a file record.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

algorithm input - a hash algorithm specifier applicable to the signature.

keyDescriptor input - a key descriptor record.

msgInFile input - a message stored in a file, representing the message belonging to the signature.

signature input - the signature specification which applies to the message.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature is a match for the specified message.
- ERR_INVALID_SIGNATURE - specified signature does not match the specified message.
- ERR_UNSUPPORTED_HASH_TYPE - the specified hash algorithm is not supported.
- ERR_UNSUPPORTED_KEY_TYPE - the specified key type is not supported.

Prospective Clients:

External Apps.

5.1.4.20 **CertMgrResult** CertMgrVerifyMbufSigWithCert (char * *certificateType*, CertMgrMemBuff * *certificate*, CertMgrMemBuff * *message*, CertMgrMemBuff * *signature*)

Uses a security certificate to verify that the specified signature matches a specified message stored in memory. The certificate determines the signature algorithm and the public key to decode the signature.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateType input - the certificate type.

certificate input - a security certificate of any type, used to verify the message signature.

message input - the message to which the signature to be verified applies.

signature input - the signature specification which applies to the message.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature is a match for the specified message.
- ERR_INVALID_SIGNATURE - specified signature does not match the specified message.
- ERR_INVALID_CERTIFICATE - unknown or incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.1.4.21 CertMgrResult CertMgrVerifyMemBufferSignature (CertMgrSigType *algorithm*, CertMgrKeyDescr * *keyDescriptor*, CertMgrMemBuff * *message*, CertMgrMemBuff * *signature*)

Uses a hash type specification and a key descriptor to verify that a specified signature matches the specified message stored in a memory buffer.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

algorithm input - a hash algorithm specifier applicable to the signature.

keyDescriptor input - a key descriptor record.

message input - a message stored in memory, representing the message belonging to the signature.

signature input - the signature specification which applies to the message.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the specified signature is a match for the specified message.
- ERR_INVALID_SIGNATURE - specified signature does not match the specified message.
- ERR_UNSUPPORTED_HASH_TYPE - the specified hash algorithm is not supported.
- ERR_UNSUPPORTED_KEY_TYPE - the specified key type is not supported.

Prospective Clients:

External Apps.

5.2 CertMgrCertProcessing.h File Reference

5.2.1 Detailed Description

This file specifies the certificate-processing plug in component for certificate manager. This plug in is responsible for verifying and parsing of certificates.

```
#include "CertMgr.h"
```

Defines

- `#define ERR_CMPROC_INVALID_CERTIFICATE ( (INT32)(CERTPROCESS_START_`
`ERRCODE - 0) )`
function return error code indicating an incorrectly formatted certificate was specified
- `#define ERR_CMPROC_UNKNOWN_CERTIFICATE ( (INT32)(CERTPROCESS_START_`
`ERRCODE - 1) )`
function return error code indicating an unknown/unsupported certificate was specified

Functions

- `CertMgrResult CertMgrCertIsMyCertificate (CertMgrMemBuff *certificate, char *certificate-`
`Type)`
This function determines the type of a certificate.
- `CertMgrResult CertMgrCertProcessingDecodeCert (CertMgrMemBuff *certificate, CertMgrCert-`
`Descriptor *certificateDescriptor)`
This function will assemble a certificate record from a raw certificate.
- `CertMgrResult CertMgrCertProcessingRelCrtDta (CertMgrCertDescriptor *certificateDescriptor)`
Releases certificate field records allocated in a previous call to CertMgrCertProcessingDecodeCert.
- `CertMgrResult CertMgrCertProcessingMatchField (CertMgrMemBuff *certificate, CertMgrCert-`
`FieldDesc *targetField, BOOL *match)`
This function will try to match a specified field with a certificate in raw form.
- `CertMgrResult CertMgrCertProcessingCertMatch (CertMgrMemBuff *certificate1, CertMgrMem-`
`Buff *certificate2, BOOL *match)`
This function will compare two raw certificates to determine if they are duplicates (do they match each other).

5.2.2 Define Documentation

5.2.2.1 `#define ERR_CMPROC_INVALID_CERTIFICATE ( (INT32)(CERTPROCESS_` `START_ERRCODE - 0) )`

function return error code indicating an incorrectly formatted certificate was specified

5.2.2.2 `#define ERR_CMPROC_UNKNOWN_CERTIFICATE ( (INT32)(CERTPROCESS_ - START_ERRCODE - 1) )`

function return error code indicating an unknown/unsupported certificate was specified

5.2.3 Function Documentation

5.2.3.1 `CertMgrResult CertMgrCertIsMyCertificate (CertMgrMemBuff * certificate, char * certificateType)`

This function determines the type of a certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes: n/a

Warning:

n/a

Parameters:

certificate Input - a security certificate in raw data.
certificateType Output - the certificate type identifier.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate type could be determined.
- ERR_CMPROC_UNKNOWN_CERTIFICATE - unknown/unsupported certificate.
- ERR_CMPROC_INVALID_CERTIFICATE - incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.2.3.2 `CertMgrResult CertMgrCertProcessingCertMatch (CertMgrMemBuff * certificate1, CertMgrMemBuff * certificate2, BOOL * match)`

This function will compare two raw certificates to determine if they are duplicates (do they match each other).

Sync (or) Async:

This is a Synchronous API.

Important Notes: n/a

Warning:

n/a

Parameters:

certificate1 Input - first certificate (in raw data format).

certificate2 Input - second certificate (in raw data format).

match Output - do the certificates match.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate comparison could be completed.
- ERR_CMPROC_UNKNOWN_CERTIFICATE - unknown/unsupported certificate.
- ERR_CMPROC_INVALID_CERTIFICATE - incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.2.3.3 **CertMgrResult** CertMgrCertProcessingDecodeCert (**CertMgrMemBuff** * *certificate*, **CertMgrCertDescriptor** * *certificateDescriptor*)

This function will assemble a certificate record from a raw certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: the memory for the certificate field records is allocated by this function, to release the allocated memory use CertMgrCertProcessingRelCrtDta.

Warning:

n/a

Parameters:

certificate Input - a security certificate in raw data.

certificateDescriptor Output - The decoded certificate structure, which contains certificate information.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate is decoded.
- ERR_CMPROC_UNKNOWN_CERTIFICATE - unknown/unsupported certificate.
- ERR_CMPROC_INVALID_CERTIFICATE - incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.2.3.4 **CertMgrResult** CertMgrCertProcessingMatchField (**CertMgrMemBuff** * *certificate*, **CertMgrCertFieldDesc** * *targetField*, **BOOL** * *match*)

This function will try to match a specified field with a certificate in raw form.

Sync (or) Async:

This is a Synchronous API.

Important Notes: n/a

Warning:

n/a

Parameters:

certificate Input - a security certificate in raw data.

targetField Input - the field specification for which a match in the certificate is desired.

match Output - a boolean, if true indicates that the specified field is a match in the certificate.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating that the certificate is supported and was properly formatted.
- ERR_CMPROC_UNKNOWN_CERTIFICATE - unknown/unsupported certificate.
- ERR_CMPROC_INVALID_CERTIFICATE - incorrectly formatted certificate.

Prospective Clients:

External Apps.

5.2.3.5 **CertMgrResult** CertMgrCertProcessingRelCrtDta (**CertMgrCertDescriptor** * *certificateDescriptor*)

Releases certificate field records allocated in a previous call to CertMgrCertProcessingDecodeCert.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateDescriptor Input - The decoded certificate structure, for which the fields should be deallocated.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - indicating successful memory release.

Prospective Clients:

External Apps.

5.3 CertMgrCryptography.h File Reference

5.3.1 Detailed Description

This file specifies the crypto graphic services supplied by the cryptographic plug in component for the certificate manager.

```
#include "CertMgr.h"
```

Defines

- `#define ERR_CMCRYPT_NO_MEMORY ( (INT32)(CERTCRYPTO_START_ERRCODE - 0) )`
function return error code indicating no memory available
- `#define ERR_CMCRYPT_UNKNOWN_HASH_TYPE ( (INT32)(CERTCRYPTO_START_ERRCODE - 1) )`
function return error code indicating an unknown hash type was specified
- `#define ERR_CMCRYPT_BUFF_TOO_SMALL ( (INT32)(CERTCRYPTO_START_ERRCODE - 2) )`
function return error code indicating supplied buffer is to small
- `#define ERR_CMCRYPT_ALGO_NOT_SUPPORTED ( (INT32)(CERTCRYPTO_START_ERRCODE - 3) )`
function return error code indicating the encryption algorithm is not supported

Functions

- `CertMgrResult CertMgrCryptoGetHashBuffSize (CertMgrSigType hashType, UINT32 *size)`
This function is used to determine the size of a hash for a particular hash algorithm type.
- `CertMgrResult CertMgrCryptoInitHash (CertMgrSigType hashType, void **context)`
This function is used to allocate and initialize a hash processing context record for the specified algorithm. This function is part of a triplet of functions (initialize, process and finalise), which are used together to generate a message hash for the specified hash type.
- `CertMgrResult CertMgrCryptoProcessHash (void *context, CertMgrMemBuff *message)`
This function is used to generate a message hash over arbitrarily long messages. This function is part of a triplet of functions (initialize, process and finalise), which are used together to generate a message hash for the specified hash type.
- `CertMgrResult CertMgrCryptoFinaliseHash (void *context, CertMgrMemBuff *message, CertMgrMemBuff *hash)`
This function is invoked at the end of a sequence of message hash processing calls to finalise the hash computation. The function generates the message hash and de-allocates the context record.
- `void CertMgrCryptoGenerateHash (CertMgrSigType hashType, CertMgrMemBuff *message, CertMgrMemBuff *hash)`
This function is used to generate a hash value for the specified hash type over a single message buffer.

- void `CertMgrCryptoDecrypt` (`CertMgrCipherAlgo` algorithm, `CertMgrKeyDescr` *key, `CertMgrMemBuff` *message, `CertMgrMemBuff` *plainText)

This function is used to decrypt the specified message with the specified key and decryption algorithm.

- `CertMgrResult` `CertMgrCryptoEncrypt` (void *keyId, `CertMgrCipherAlgo` algorithm, `CertMgrMemBuff` *plainText, `CertMgrMemBuff` *message)

This function is used to encrypt the specified plain text with the specified key and encryption algorithm.

5.3.2 Define Documentation

5.3.2.1 #define ERR_CMCRYPT_ALGO_NOT_SUPPORTED ((INT32)(CERTCRYPTO_START_ERRCODE - 3))

function return error code indicating the encryption algorithm is not supported

5.3.2.2 #define ERR_CMCRYPT_BUFF_TOO_SMALL ((INT32)(CERTCRYPTO_START_ERRCODE - 2))

function return error code indicating supplied buffer is too small

5.3.2.3 #define ERR_CMCRYPT_NO_MEMORY ((INT32)(CERTCRYPTO_START_ERRCODE - 0))

function return error code indicating no memory available

5.3.2.4 #define ERR_CMCRYPT_UNKNOWN_HASH_TYPE ((INT32)(CERTCRYPTO_START_ERRCODE - 1))

function return error code indicating an unknown hash type was specified

5.3.3 Function Documentation

5.3.3.1 void `CertMgrCryptoDecrypt` (`CertMgrCipherAlgo` algorithm, `CertMgrKeyDescr` *key, `CertMgrMemBuff` *message, `CertMgrMemBuff` *plainText)

This function is used to decrypt the specified message with the specified key and decryption algorithm.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

algorithm input - the decryption algorithm to use.

key input - a public key descriptor.

message input - the encrypted message.

plainText output - the decrypted message.

Returns:

Return Type (void)

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
CertMgrResult    result;
CertMgrMemBuff   plainText;
CertMgrKeyDescr  key,

... initialize (plainText, key etc.)

CertMgrCryptoDecrypt (CERTMGR_CIPHER_RSA, &key, message, &plainText );
...
```

5.3.3.2 **CertMgrResult** CertMgrCryptoEncrypt (void * *keyId*, **CertMgrCipherAlgo** *algorithm*, **CertMgrMemBuff** * *plainText*, **CertMgrMemBuff** * *message*)

This function is used to encrypt the specified plain text with the specified key and encryption algorithm.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

keyId input - a key identifier for secure key data access. NOTE: The format of keyId is dependent on the specific platform-dependent implementation of secure key handling used by the Cryptography plug-in.

algorithm input - the encryption algorithm to be used.

plainText input - the plain text message.

message output - the encrypted message.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - message was successfully encrypted

Return Type (void)

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```

...
CertMgrResult      result;
CertMgrMemBuff     message;
void               *keyId;

... initialize (message etc.)

CertMgrCryptoEncrypt(keyId, CERTMGR_CIPHER_RSA, plainText, &message );
...

```

5.3.3.3 **CertMgrResult** CertMgrCryptoFinaliseHash (void * *context*, **CertMgrMemBuff** * *message*, **CertMgrMemBuff** * *hash*)

This function is invoked at the end of a sequence of message hash processing calls to finalise the hash computation. The function generates the message hash and de-allocates the context record.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

context input - The address of a context record specified as a pointer to void.

message input - a message specification record.

hash output - a buffer, which will receive the computed hash.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the hash was correctly finalised.
- ERR_CMCRYPT_BUFF_TOO_SMALL - the supplied hash buffer is too small.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```

...
void               *hashContext;
CertMgrMemBuff     *messageBuffer;
CertMgrSigType     hashType = CERTMGR_HASH_MD2_RSA;
CertMgrMemBuff     hash;
UINT32             hashBuffSize;

CertMgrCryptoGetHashBuffSize (hashType, &hashBuffSize);

... initialize hash buffer etc.

result = CertMgrCryptoInitHash( hashType, &hashContext );

if(result == RESULT_CODE_SUCCESS)

```

```

{
    moreInMessage = getMessageBuffer(targetMessage, messageBuffer)
    while (moreInMessage == TRUE)
    {
        CertMgrCryptoProcessHash(hashContext, messageBuffer);
        moreInMessage = getMessageBuffer(targetMessage, messageBuffer);
    }
    CertMgrCryptoFinaliseHash(hashContext, &hash)
    ...
}

```

5.3.3.4 void CertMgrCryptoGenerateHash (CertMgrSigType hashType, CertMgrMemBuff * message, CertMgrMemBuff * hash)

This function is used to generate a hash value for the specified hash type over a single message buffer.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

hashType input - the desired hash type.

message input - a message specification record.

hash output - a buffer receiving the computed hash.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the hash was correctly computed.
- ERR_CMCRYPT_UNKNOWN_HASH_TYPE - the specified hash type is not supported.
- ERR_CMCRYPT_BUFF_TOO_SMALL - the supplied hash buffer was too small.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```

...
CertMgrSigType      hashType = CERTMGR_HASH_MD2_RSA;
CertMgrMemBuff      hash;
UINT32              hashBuffSize;
CertMgrMemBuff      *messageBuffer;

CertMgrCryptoGetHashBuffSize (hashType, &hashBuffSize);

... initialize hash buffer etc.

result = CertMgrCryptoGenerateHash( hashType, message, &hash );

if (result == RESULT_CODE_SUCCESS)
{
    ...
}

```

5.3.3.5 **CertMgrResult** CertMgrCryptoGetHashBuffSize (**CertMgrSigType** *hashType*, **UINT32** **size*)

This function is used to determine the size of a hash for a particular hash algorithm type.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

hashType input - The type of hash algorithm for which we want the hash size.

size output - The size of the hash in bytes.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the component was successfully initialized.
- ERR_CMCRYPT_UNKNOWN_HASH_TYPE - the specified hash type was unknown.

Prospective Clients:

Certificate manager foundation API.

5.3.3.6 **CertMgrResult** CertMgrCryptoInitHash (**CertMgrSigType** *hashType*, **void **** *context*)

This function is used to allocate and initialize a hash processing context record for the specified algorithm.

This function is part of a triplet of functions (initialize, process and finalise), which are used together to generate a message hash for the specified hash type.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

hashType input - The hash type specification.

context output - The address of a pointer to void, which will receive the context record address.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the context record was properly allocated and initialized.
- ERR_CMCRYPT_NO_MEMORY - no memory to allocate a context record.

- ERR_CMCRYPT_UNKNOWN_HASH_TYPE - the hash type is not supported.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
void                *hashContext;
CertMgrMemBuff      *messageBuffer;
CertMgrSigType       hashType = CERTMGR_HASH_MD2_RSA;
CertMgrMemBuff       hash;
UINT32              hashBuffSize;

CertMgrCryptoGetHashBuffSize (hashType, &hashBuffSize);

... initialize hash buffer etc.

result = CertMgrCryptoInitHash( hashType, &hashContext );

if(result == RESULT_CODE_SUCCESS)
{
    moreInMessage = getMessageBuffer(targetMessage, messageBuffer)
    while(moreInMessage == TRUE)
    {
        CertMgrCryptoProcessHash(hashContext, messageBuffer);
        moreInMessage = getMessageBuffer(targetMessage, messageBuffer);
    }
    CertMgrCryptoFinaliseHash(hashContext, &hash)
}
...
```

5.3.3.7 CertMgrResult CertMgrCryptoProcessHash (void * context, CertMgrMemBuff * message)

This function is used to generate a message hash over arbitrarily long messages. This function is part of a triplet of functions (initialize, process and finalise), which are used together to generate a message hash for the specified hash type.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: if the message fits into a single buffer, we use the other supplied functions for this (CertMgrCryptoGenerateHash)

Parameters:

context input - The address of a context record.

message input - a message specification record.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the hash was correctly computed over the specified message block

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
void                               *hashContext;
CertMgrMemBuff                     *messageBuffer;
CertMgrSigType                     hashType = CERTMGR_HASH_MD2_RSA;
CertMgrMemBuff                     hash;
UINT32                             hashBuffSize;

CertMgrCryptoGetHashBuffSize (hashType, &hashBuffSize);

... initialize hash buffer etc.

result = CertMgrCryptoInitHash( hashType, &hashContext );

if(result == RESULT_CODE_SUCCESS)
{
    moreInMessage = getMessageBuffer(targetMessage, messageBuffer)
    while(moreInMessage == TRUE)
    {
        CertMgrCryptoProcessHash(hashContext, messageBuffer);
        moreInMessage = getMessageBuffer(targetMessage, messageBuffer);
    }
    CertMgrCryptoFinaliseHash(hashContext, &hash)
}
...
```

5.4 CertMgrDebug.h File Reference

5.4.1 Detailed Description

This is the header file for the LiMo Certificate Manager Foundation API.

Defines

- #define **CM_LOG** 1
- #define **CERTMGR_DEBUG**(fmt, args...)

5.4.2 Define Documentation

5.4.2.1 #define CERTMGR_DEBUG(fmt, args...)

Value:

```
printf("Func %s(on line %d of file %s): "fmt"\n", \
      __FUNCTION__, __LINE__, __FILE__, ##args)
```

5.5 CertMgrInternal.h File Reference

5.5.1 Detailed Description

This is the header file for the LiMo Certificate Manager Foundation API.

```
#include "CertMgr.h"
#include "CertMgrSignature.h"
#include "CertMgrRevocationStatus.h"
```

Data Structures

- struct `_SignatureInitParas`
- struct `_CertList`

Defines

- #define `CIPHER_PADDING` 1
- #define `MAX_NAME_LENGTH` 256
- #define `MAX_TYPE_LENGTH` 256
- #define `MAX_PUBKEY_LENGTH` (2*1024)
- #define `MAX_SIG_ALGO_LENGTH` 32
- #define `MAX_CERT_TYPE_LENGTH` 10
- #define `HASH_MD2_LENGTH` 16
- #define `HASH_MD5_LENGTH` 16
- #define `HASH_SHA1_LENGTH` 20
- #define `HASH_MAX_LENGTH` 20
- #define `MAX_STORE_ID_LENGTH` 20
- #define `MAX_STORE_TYPE_LENGTH` 20
- #define `MAX_TYPE_SIZE` 50
- #define `MAX_ID_SIZE` 50
- #define `MAX_DATA_SIZE` 256
- #define `MAX_UINT32_STRLEN` 11
- #define `FLAG_ISSUER` 0
- #define `FLAG_SUBJECT` 1
- #define `CERTMGR_X509_TYPE` "X509"
- #define `METADATA_TYPELEN_SIZE` (sizeof(`UINT32`))
- #define `METADATA_OWNERLEN_SIZE` (sizeof(`UINT32`))
- #define `METADATA_LENGTH_SIZE` (sizeof(`UINT32`))
- #define `MAX_BUFFER_SIZE` (4*1024)
- #define `METADATA_PATH` "/"
- #define `ERR_INPUT_PARAMETER` ((`INT32`)(`CERTMGR_START_ERRCODE` - 16))
- #define `ERR_UNKNOWN_CERTIFICATE` ((`INT32`)(`CERTMGR_START_ERRCODE` - 17))
- #define `ERR_CERTIFICATE_FORMAT_ERROR` ((`INT32`)(`CERTMGR_START_ERRCODE` - 18))
- #define `ERR_CMPROC_UNSUPPORTED_SIG_ALGO` ((`INT32`)(`CERTMGR_START_ERRCODE` - 19))
- #define `ERR_NOT_IMPLEMENTED` ((`INT32`)(`CERTMGR_START_ERRCODE` - 20))

- #define **ERR_EXTERN_CHAIN_IS_EMPTY** ((INT32)(CERTMGR_START_ERRCODE - 21))
- #define **ERR_DIGEST_INIT** ((INT32)(CERTMGR_START_ERRCODE - 22))
- #define **ERR_DIGEST_UPDATE** ((INT32)(CERTMGR_START_ERRCODE - 23))
- #define **ERR_DIGEST_FINAL** ((INT32)(CERTMGR_START_ERRCODE - 24))
- #define **ERR_NO_TARGET_FIELD** ((INT32)(CERTMGR_START_ERRCODE - 25))
- #define **ERR_PLUGIN_SYM** ((INT32)(CERTMGR_START_ERRCODE - 26))
- #define **ERR_INVALID_ROOT_CERTIFICATE** ((INT32)(CERTMGR_START_ERRCODE - 27))
- #define **ERR_PLUGIN_IS_UNAVAILABLE** ((INT32)(CERTMGR_START_ERRCODE - 28))
- #define **ERR_FILE_NOT_EXIST** ((INT32)(CERTMGR_START_ERRCODE - 29))
- #define **ERR_INDEX_NOT_FOUND** ((INT32)(CERTMGR_START_ERRCODE - 30))
- #define **ERR_DUPLICATE_NODE** ((INT32)(CERTMGR_START_ERRCODE - 31))
- #define **ERR_INVALID_ID** ((INT32)(CERTMGR_START_ERRCODE - 32))
- #define **OPERATION_SUCCESS** ((UINT32)0)
- #define **ERR_CMSTORE_IS_FULL** ((INT32)(CERTSTORE_START_ERRCODE - 1))
- #define **ERR_FILE_OPEN** ((UINT32)(CERTSTORE_START_ERRCODE - 6))
- #define **ERR_CERT_NOFIND** ((UINT32)(CERTSTORE_START_ERRCODE - 7))
- #define **ERR_MEM_MALLOC** ((UINT32)(CERTSTORE_START_ERRCODE - 8))
- #define **ERR_MAX_CERTID** ((UINT32)(CERTSTORE_START_ERRCODE - 11))
- #define **ERR_SYSTEM_CALL** ((UINT32)(CERTSTORE_START_ERRCODE - 12))
- #define **ERR_FILE_CREATE** ((UINT32)(CERTSTORE_START_ERRCODE - 13))
- #define **ERR_UNSUPPORTED_CERT_TYPE** ((UINT32)(CERTSTORE_START_ERRCODE - 14))
- #define **ERR_INVALID_METADATA_ID** ((INT32)(CERTMETADATA_START_ERRCODE - 10))
- #define **ERR_UNSUPPORTED_METADATA** ((INT32)(CERTMETADATA_START_ERRCODE - 11))
- #define **ERR_CMSIG_DECRYPT_FAILED** ((INT32)(CERTSIGNATURE_START_ERRCODE - 2))

Typedefs

- typedef _SignatureInitParas **SignatureInitParas**
- typedef _CertList **CertList**
- typedef **CertMgrResult**(*) **CertMgrCryptoInitHash_t** (**CertMgrSigType** hashType, void **context)
- typedef **CertMgrResult**(*) **CertMgrCryptoProcessHash_t** (void *context, **CertMgrMemBuff** *message)
- typedef **CertMgrResult**(*) **CertMgrCryptoFinaliseHash_t** (void *context, **CertMgrMemBuff** *message, **CertMgrMemBuff** *hash)
- typedef void(*) **CertMgrCryptoDecrypt_t** (**CertMgrCipherAlgo** algorithm, **CertMgrKeyDescr** *key, **CertMgrMemBuff** *message, **CertMgrMemBuff** *plainText)
- typedef **CertMgrResult**(*) **CertMgrSignatureInitContext_t** (**CertMgrSigType** algorithm, void **context)
- typedef **CertMgrResult**(*) **CertMgrSignatureVerify_t** (void *context, **CertMgrKeyDescr** *key-Descriptor, **CertMgrMemBuff** *message, **BOOL** final, **CertMgrMemBuff** *signature)
- typedef void(*) **CertMgrSignatureReleaseContext_t** (void *context)
- typedef **CertMgrResult**(*) **CertMgrSignatureLoadCryptoInfo_t** (**CertMgrPlugInDescr** *crypto-PlugIns, **UINT32** noOfPlugIns)

- typedef [CertMgrResult](#)(*) [CertMgrSignatureInitialize_t](#) ([CertMgrSigType](#) algorithm, char *initParas)
- typedef [CertMgrResult](#)(*) [CertMgrCertIsMyCertificate_t](#) ([CertMgrMemBuff](#) *certificate, char *certificateType)
- typedef [CertMgrResult](#)(*) [CertMgrCertProcessingDecodeCert_t](#) ([CertMgrMemBuff](#) *certificate, [CertMgrCertDescriptor](#) *certificateDescriptor)
- typedef [CertMgrResult](#)(*) [CertMgrCertProcessingRelCrtDta_t](#) ([CertMgrCertDescriptor](#) *certificateDescriptor)
- typedef [CertMgrResult](#)(*) [CertMgrCertProcessingMatchField_t](#) ([CertMgrMemBuff](#) *certificate, [CertMgrCertFieldDesc](#) *targetField, [BOOL](#) *match)
- typedef [CertMgrResult](#)(*) [CertMgrCertProcessingCertMatch_t](#) ([CertMgrMemBuff](#) *certificate1, [CertMgrMemBuff](#) *certificate2, [BOOL](#) *match)
- typedef [CertMgrResult](#)(*) [StoreInitializePtr](#) ([CertMgrStorageId](#) storeId, char *initParas)
- typedef [CertMgrResult](#)(*) [StoreStoreCertificatePtr](#) ([CertMgrMemBuff](#) *certificate, [CertMgrCertId](#) *certificateId)
- typedef [CertMgrResult](#)(*) [StoreRemoveCertificatePtr](#) ([CertMgrCertId](#) *certificateId)
- typedef [CertMgrResult](#)(*) [StoreEnableCertificatePtr](#) ([CertMgrCertId](#) *certificateId)
- typedef [CertMgrResult](#)(*) [StoreDisableCertificatePtr](#) ([CertMgrCertId](#) *certificateId)
- typedef [CertMgrResult](#)(*) [StoreRemoveAllUserCertificatesPtr](#) (void)
- typedef [CertMgrResult](#)(*) [StoreRetrieveCertificatePtr](#) ([CertMgrCtx](#) *context, [CertMgrMemBuff](#) *certificate, [CertMgrCertId](#) *certificateId)
- typedef [CertMgrResult](#)(*) [StoreRetrieveCertificateStatusPtr](#) ([CertMgrCertId](#) *certificateId, [CertMgrCertStatus](#) *status)
- typedef [CertMgrResult](#)(*) [CertMgrRevocationStatusVerify_t](#) ([CertMgrCertDescriptor](#) *, enum [CertMgrRevocationStatus](#) *)

5.6 CertMgrPlugin.h File Reference

5.6.1 Detailed Description

This is the header file for LiMo Certificate Manager Plug-in framework API.

```
#include "CertMgrInternal.h"
```

Data Structures

- struct **_CertMgrPlugin**
- struct **_CertMgrPluginType**

Typedefs

- typedef **_CertMgrPlugin** **CertMgrPlugin**
- typedef **_CertMgrPluginType** **CertMgrPluginType**

Enumerations

- enum **CertMgrPlugInType** {
 CERTMGR_STORE, **CERTMGR_DIGITALSIGNATURE**, **CERTMGR_CRYPTOGRAPHY**,
 CERTMGR_STATUS,
 CERTMGR_PROCESSING }

Functions

- **CertMgrPlugin** * **CertMgrFindPlugin** (**CertMgrPlugInType** type, char *id)
- **CertMgrPlugin** * **CertMgrFindNextPlugin** (**CertMgrPlugin** *current)
- void **CertMgrInit** (void)
- void **CertMgrExit** (void)

5.6.2 Enumeration Type Documentation

5.6.2.1 enum **CertMgrPlugInType**

Different plug-in types

Enumerator:

CERTMGR_STORE Certificate Store Plug-in
CERTMGR_DIGITALSIGNATURE Digital signature plug-in
CERTMGR_CRYPTOGRAPHY Certificate cryptography plug-in
CERTMGR_STATUS Certificate revocation plug-in
CERTMGR_PROCESSING Certificate processing plug-in

5.6.3 Function Documentation

5.6.3.1 void CertMgrExit (void)

Release all resources used for plug-ins.

5.6.3.2 CertMgrPlugin* CertMgrFindNextPlugin (CertMgrPlugin * *current*)

Retrives the next plugin in the same plugin type.

5.6.3.3 CertMgrPlugin* CertMgrFindPlugin ([CertMgrPlugInType](#) *type*, char * *id*)

Retrives the plugin by type and ID.

Parameters:

type -plugin type, for example, "Store", "DigitalSignature".

id -plugin id. for example "md5", "RSA". For store plugin, if the id is null, then returns the default plugin.

5.6.3.4 void CertMgrInit (void)

Initilize all the plug-ins.

5.7 CertMgrRevocationStatus.h File Reference

5.7.1 Detailed Description

This file specifies the revocation status plug in componenet for certificate manager. This plug in is responsible for handling certificate status request.

```
#include "CertMgr.h"
```

Defines

- `#define ERR_CMREVOC_NO_REVOC_STATUS ( (INT32)(CERTREVOC_STATUS_START - ERRCODE - 0) )`
function return error code indicating no revocation status is available for the specified certificate
- `#define ERR_CMREVOC_INVALID_CERTIFICATE ( (INT32)(CERTREVOC_STATUS_START - ERRCODE - 1) )`
function return error code indicating the specified certificate is invalid

Enumerations

- `enum CertMgrRevocationStatus { CERTMGR_NOT_REVOKED, CERTMGR_REVOKED }`

Functions

- `CertMgrResult CertMgrRevocationStatusVerify (CertMgrCertDescriptor *certificateDescriptor, enum CertMgrRevocationStatus *status)`
This function will obtain the current revocation status of a certificate.

5.7.2 Define Documentation

5.7.2.1 `#define ERR_CMREVOC_INVALID_CERTIFICATE ( (INT32)(CERTREVOC_STATUS_START - ERRCODE - 1) )`

function return error code indicating the specified certificate is invalid

5.7.2.2 `#define ERR_CMREVOC_NO_REVOC_STATUS ( (INT32)(CERTREVOC_STATUS_START - ERRCODE - 0) )`

function return error code indicating no revocation status is available for the specified certificate

5.7.3 Enumeration Type Documentation

5.7.3.1 `enum CertMgrRevocationStatus`

The revocation status

Enumerator:

CERTMGR_NOT_REVOKED the certificate has not been revoked

CERTMGR_REVOKED the certificate has been revoked

5.7.4 Function Documentation**5.7.4.1 *CertMgrResult* CertMgrRevocationStatusVerify (*CertMgrCertDescriptor* * *certificateDescriptor*, enum *CertMgrRevocationStatus* * *status*)**

This function will obtain the current revocation status of a certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Warning:

n/a

Parameters:

certificateDescriptor Input - The decoded certificate structure, which contains certificate information.

status Output - Status indicating whether the certificate is revoked

Returns:

Return Type (*CertMgrResult*)

- OPERATION_SUCCESS - Indicating revocation status could be established
- ERR_CMREVOC_NO_REVOC_STATUS - Indicating that the revocation status could not be established
- ERR_CMREVOC_INVALID_CERTIFICATE - Unknown/unsupported certificate.

Prospective Clients:

External Apps.

5.8 CertMgrSignature.h File Reference

5.8.1 Detailed Description

This is the component which implements digital signature capability for the certificate manager.

```
#include "CertMgr.h"
```

Data Structures

- struct [CertMgrPlugInDescr](#)

Defines

- #define [ERR_CMSIG_NO_MEMORY](#) ((INT32)(CERTSIGNATURE_START_ERRCODE - 0))
function return error code indicating no memory available
- #define [ERR_CMSIG_UNSUPPORTED_HASH_TYP](#) ((INT32)(CERTSIGNATURE_START_ERRCODE - 1))
function return error code indicating the specified hash type is not supported

Functions

- [CertMgrResult](#) [CertMgrSignatureLoadCryptoInfo](#) ([CertMgrPlugInDescr](#) *cryptoPlugIns, [UINT32](#) noOfPlugIns)
This function is used to inform the signature module, where it can find the various crypto plug-ins.
- [CertMgrResult](#) [CertMgrSignatureInitContext](#) ([CertMgrSigType](#) algorithm, void **context)
This function is used to allocate and initialize a context record, used for signature verification and generation, whenever a message is too big to fit into a single processing buffer. The function is the mirror function to [CertMgrSignatureReleaseContext](#), which releases the context. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:
 - allocate and initialize context
 - verify or generate (repeated n times)
 - release context.
- void [CertMgrSignatureReleaseContext](#) (void *context)
This function is used to de-allocate a context record, used for signature verification and generation, whenever a message is too big to fit into a single processing buffer and the verifying or generating function must be called multiple times. The function is the mirror function to [CertMgrSignatureInitContext](#), which allocates and initializes the context. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:
 - allocate and initialize context
 - verify or generate (repeated n times)
 - release context.
- [CertMgrResult](#) [CertMgrSignatureVerify](#) (void *context, [CertMgrKeyDescr](#) *keyDescriptor, [CertMgrMemBuff](#) *message, [BOOL](#) final, [CertMgrMemBuff](#) *signature)

This function is used to verify that a specified message has the specified signature based on the specified key and digest algorithm. Note, the digest algorithm is being specified when the context is allocated with the CertMgrSignatureInitContext function. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

- **CertMgrResult CertMgrSignatureGenerate** (void *context, void *keyId, CertMgrMemBuff *message, **BOOL** final, CertMgrMemBuff *signature)

This function is used to generate a signature over an arbitrarily long message. The signature is based on some key identifier and a digest algorithm. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

- **CertMgrResult CertMgrSignatureInitialize** (CertMgrSigType algorithm, char *initParas)

This function is used to initialize the signature component.

5.8.2 Define Documentation

5.8.2.1 #define ERR_CMSIG_NO_MEMORY ((INT32)(CERTSIGNATURE_START - ERRCODE - 0))

function return error code indicating no memory available

5.8.2.2 #define ERR_CMSIG_UNSUPPORTED_HASH_TYP ((INT32)(CERTSIGNATURE_START - ERRCODE - 1))

function return error code indicating the specified hash type is not supported

5.8.3 Function Documentation

5.8.3.1 **CertMgrResult CertMgrSignatureGenerate** (void * context, void * keyId, CertMgrMemBuff * message, **BOOL** final, CertMgrMemBuff * signature)

This function is used to generate a signature over an arbitrarily long message. The signature is based on some key identifier and a digest algorithm. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: The digest algorithm is being specified during context allocation
- Note: The key identifier must allow secure handling of the signing key.
- Note: Handling of private signing keys depends on the local security set up.

Parameters:

context input/output - a context record address previously allocated with a call to CertMgrSignature-InitContext.

keyId input - a key identifier for secure key data access.

message input - a message specification record (either the whole message or a fragment).

final input - a boolean, if true, signals that the current invocation contains the last message part in its message buffer.

signature input - a signature descriptor record.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the signature is correct for the specified message.
- ERR_CMSIG_UNSUPPORTED_HASH_TYP - the specified algorithm is not supported.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
CertMgrResult    result;
void             *context;
int              iteration;
CertMgrFilBuff  *msgOnFile
CertMgrMemBuff  *message;
...
... some initialization etc.
...
result = CertMgrSignatureInitContext(CERTMGR_HASH_MD2, &context );

if(result == OPERATION_SUCCESS)
{
    while( allDone == FALSE && moreBufs == TRUE )
    {
        moreBufs = getNextMessageBuffer(msgOnFile, message, ...)
        CertMgrSignatureGenerate(context, keyDescriptor, message, FALSE, signature)
    }
    CertMgrSignatureGenerate(context, keyDescriptor, message, TRUE, signature)
    CertMgrSignatureReleaseContext(context)
    ...
    ...
}
```

5.8.3.2 **CertMgrResult** CertMgrSignatureInitContext (**CertMgrSigType** *algorithm*, void ***context*)

This function is used to allocate and initialize a context record, used for signature verification and generation, whenever a message is too big to fit into a single processing buffer. The function is the mirror function to CertMgrSignatureReleaseContext, which releases the context. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

algorithm input - specifies the digest algorithm for which the signature context is used.

context output - address of variable, receiving the context address.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the context was successfully allocated and initialized.
- ERR_CMSIG_NO_MEMORY - the context record could not be allocated.
- ERR_CMSIG_UNSUPPORTED_HASH_TYP - the specified algorithm is not supported.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
CertMgrResult  result;
void           *context;

result = CertMgrSignatureInitContext(CERTMGR_HASH_MD2, &context );

if(result == OPERATION_SUCCESS)
...

```

5.8.3.3 CertMgrResult CertMgrSignatureInitialize (CertMgrSigType algorithm, char * initParas)

This function is used to initialize the signature component.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

Note: n/a

Parameters:

algorithm input - specifies the signature type.

initParas input - contains initialization string which is used to connect to proper cryptography plug-in.

Returns:

Return Type (RESULT_CODE_T)

OPERATION_SUCCESS - the component was successfully initialized.

Prospective Clients:

Certificate manager foundation API.

5.8.3.4 **CertMgrResult** CertMgrSignatureLoadCryptoInfo (**CertMgrPlugInDescr** * *cryptoPlugIns*, **UINT32** *noOfPlugIns*)

This function is used to inform the signature module, where it can find the various crypto plug-ins.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

cryptoPlugIns input - an array of cryptographic plug-in descriptors needed to access the cryptographic functions of the CM.

noOfPlugIns input - the number of plug-in descriptors in the array.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the plug-in descriptors were successfully loaded.

Prospective Clients:

Certificate manager foundation API.

5.8.3.5 **void** CertMgrSignatureReleaseContext (**void** * *context*)

This function is used to de-allocate a context record, used for signature verification and generation, whenever a message is too big to fit into a single processing buffer and the verifying or generating function must be called multiple times. The function is the mirror function to CertMgrSignatureInitContext, which allocates and initializes the context. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

context input - the context to be released.

Returns:

Return Type (void)

Prospective Clients:

Certificate manager foundation API.

5.8.3.6 CertMgrResult CertMgrSignatureVerify (void * context, CertMgrKeyDescr * keyDescriptor, CertMgrMemBuff * message, BOOL final, CertMgrMemBuff * signature)

This function is used to verify that a specified message has the specified signature based on the specified key and digest algorithm. Note, the digest algorithm is being specified when the context is allocated with the CertMgrSignatureInitContext function. The process of verification or generation of a signature over an arbitrarily long message, follows a sequence laid out below:

- allocate and initialize context
- verify or generate (repeated n times)
- release context.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

context input/output - a context record address previously allocated with a call to CertMgrSignatureInitContext.

keyDescriptor input - a key descriptor record.

message input - a message specification record (either a fragment or the whole message).

final input - a boolean, if true, signals that the current invocation contains the last message part in its message buffer.

signature input - a signature descriptor record.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - the signature is correct for the specified message.
- ERR_CMSIG_UNSUPPORTED_HASH_TYP - the specified algorithm is not supported.

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
CertMgrResult    result;
void             *context;
int              iteration;
CertMgrFilBuff   *msgOnFile
CertMgrMemBuff   *message;
...
... some initialization etc.
...
result = CertMgrSignatureInitContext(CERTMGR_HASH_MD2, &context );

if(result == OPERATION_SUCCESS)
{
    while( allDone == FALSE && moreBufs == TRUE )
    {
        moreBufs = getNextMessageBuffer(msgOnFile, message, ...)
        CertMgrSignatureVerify(context, keyDescriptor, message, FALSE, signature)
    }
    CertMgrSignatureVerify(context, keyDescriptor, message, TRUE, signature)
    CertMgrSignatureReleaseContext(context)
    ...
    ...
}
```

5.9 CertMgrStore.h File Reference

5.9.1 Detailed Description

This file specifies the storage services supplied by the certificate store plug-in component for the certificate manager.

```
#include "CertMgr.h"
```

Defines

- `#define ERR_CMSTORE_IS_FULL ( (INT32)(CERTSTORE_START_ERRCODE - 1) )`
function return error code indicating insufficient storage space available

Functions

- `CertMgrResult CertMgrStoreInitialize (CertMgrStorageId storeId, char *initParas)`
This function is used to initialize the store.
- `CertMgrResult CertMgrStoreStoreCertificate (CertMgrMemBuff *certificate, CertMgrCertId *certificateId)`
This function is used to store one certificate to this store.
- `CertMgrResult CertMgrStoreRemoveCertificate (CertMgrCertId *certificateId)`
This function is used to remove one certificate from this store.
- `CertMgrResult CertMgrStoreEnableCertificate (CertMgrCertId *certificateId)`
This function is used to enable one certificate from this store.
- `CertMgrResult CertMgrStoreDisableCertificate (CertMgrCertId *certificateId)`
This function is used to disable one certificate from this store.
- `CertMgrResult CertMgrStoreRemoveAllUserCertificates (void)`
This function is used to remove all certificates from this store.
- `CertMgrResult CertMgrStoreRetrieveCertificate (CertMgrCtx *context, CertMgrMemBuff *certificate, CertMgrCertId *certificateId)`
This function is used to retrieve one or more certificates.
- `CertMgrResult CertMgrStoreRetrieveCertificateStatus (CertMgrCertId *certificateId, CertMgrCertStatus *status)`
This function is used to get the status (enabled/disabled) of the specified certificate.

5.9.2 Define Documentation

5.9.2.1 `#define ERR_CMSTORE_IS_FULL ( (INT32)(CERTSTORE_START_ERRCODE - 1) )`

function return error code indicating insufficient storage space available

5.9.3 Function Documentation

5.9.3.1 **CertMgrResult** CertMgrStoreDisableCertificate (**CertMgrCertId** * *certificateId*)

This function is used to disable one certificate from this store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - specifies the certificate to be disabled.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully disabled
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.2 **CertMgrResult** CertMgrStoreEnableCertificate (**CertMgrCertId** * *certificateId*)

This function is used to enable one certificate from this store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - specifies the certificate to be enabled.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully enabled
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.3 **CertMgrResult** CertMgrStoreInitialize (**CertMgrStorageId** *storeId*, char * *initParas*)

This function is used to initialize the store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

storeId input - tells the store what id it has.

initParas input - some initialization string needed by the store.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - initialization successful
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.4 **CertMgrResult** CertMgrStoreRemoveAllUserCertificates (void)

This function is used to remove all certificates from this store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - all certificates successfully removed
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.5 **CertMgrResult** CertMgrStoreRemoveCertificate (**CertMgrCertId** * *certificateId*)

This function is used to remove one certificate from this store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - specifies the certificate to be removed.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully removed
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.6 **CertMgrResult** CertMgrStoreRetrieveCertificate (**CertMgrCtx** * *context*, **CertMgrMemBuff** * *certificate*, **CertMgrCertId** * *certificateId*)

This function is used to retrieve one or more certificates.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: The context is used to keep track of the retrieval operation over successive calls
- The context must be initialized to zero during the first call of this function.
- Additionally, if the retrieval is to be by field, the field descriptor and the retrieveByField flag must be set appropriately.

Parameters:

context input/output - the address of a context record.

certificate output - specifies the certificate memory buffer retrieved.

certificateId output - specifies the certificate Id retrieved.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully retrieved
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

Example of how this function would be called:

```
...
CertMgrCtx      *context;
CertMgrMemBuff  messageBuffer,
CertMgrCertId   certificateId
CertMgrResult    result = RESULT_CODE_SUCCESS;

... initialize context, certificate buffer etc.

while( (result == RESULT_CODE_SUCCESS) && (notFound == TRUE) )
{
    result = CertMgrStoreRetrieveCertificate(context, &messageBuffer, &certificateId)
    ...
}

...
```

5.9.3.7 **CertMgrResult** CertMgrStoreRetrieveCertificateStatus (**CertMgrCertId** * *certificateId*, **CertMgrCertStatus** * *status*)

This function is used to to get the status (enabled/disabled) of the specified certificate.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificateId input - specifies the certificate to be retrieved.

status output - specifies the status (enabled/disabled) of the certificate.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate status successfully retrieved
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

5.9.3.8 **CertMgrResult** CertMgrStoreStoreCertificate (**CertMgrMemBuff** * *certificate*, **CertMgrCertId** * *certificateId*)

This function is used to store one certificate to this store.

Sync (or) Async:

This is a Synchronous API.

Important Notes:

- Note: n/a

Parameters:

certificate input - specifies the certificate (memory buffer) to be stored.

certificateId input/output - the certificate id of the certificate to be stored.

Returns:

Return Type (CertMgrResult)

- OPERATION_SUCCESS - certificate successfully stored
- ERR_CMSTORE_DUP_CERTIFICATE - certificate already exists in the store
- ERR_CMSTORE_IS_FULL - no space left in store
- ERR_CMSTORE_READ_ONLY - the store is a "read only" type
- Otherwise return one appropriate error

Prospective Clients:

Certificate manager foundation API.

Index

- [_CertMgrCertLinkedList, 11](#)
- [_CertMgrCertLinkedList](#)
 - [certificate, 11](#)
 - [next, 11](#)
- [algorithm](#)
 - [CertMgrPlugInDescr, 23](#)
- [BOOL](#)
 - [CertMgr.h, 35](#)
- [CERTCRYPTO_START_ERRCODE](#)
 - [CertMgr.h, 31](#)
- [certificate](#)
 - [_CertMgrCertLinkedList, 11](#)
- [certificateNo](#)
 - [CertMgrCertId, 15](#)
- [CERTMETADATA_START_ERRCODE](#)
 - [CertMgr.h, 31](#)
- [CertMgr.h, 25](#)
- [CertMgr.h](#)
 - [BOOL, 35](#)
 - [CERTCRYPTO_START_ERRCODE, 31](#)
 - [CERTMETADATA_START_ERRCODE, 31](#)
 - [CERTMGR_CERTFLD_EXTENSIONS, 31](#)
 - [CERTMGR_CERTFLD_ISSUER, 31](#)
 - [CERTMGR_CERTFLD_ISSUER_UID, 31](#)
 - [CERTMGR_CERTFLD_PUBLIC_KEY, 32](#)
 - [CERTMGR_CERTFLD_SERIALNO, 32](#)
 - [CERTMGR_CERTFLD_SIG_ALGO, 32](#)
 - [CERTMGR_CERTFLD_SIGNATURE, 32](#)
 - [CERTMGR_CERTFLD_SIGNED_DATA, 32](#)
 - [CERTMGR_CERTFLD_SUBJECT, 32](#)
 - [CERTMGR_CERTFLD_SUBJECT_UID, 32](#)
 - [CERTMGR_CERTFLD_VALIDITY, 32](#)
 - [CERTMGR_CERTFLD_VERSION, 32](#)
 - [CERTMGR_CERTID_HASH_SIZE, 32](#)
 - [CERTMGR_CIPHER_RSA, 32](#)
 - [CERTMGR_HASH_MD2_RSA, 33](#)
 - [CERTMGR_HASH_MD5_RSA, 33](#)
 - [CERTMGR_HASH_SHA1_RSA, 33](#)
 - [CERTMGR_INIT_CONTEXT, 33](#)
 - [CERTMGR_START_ERRCODE, 33](#)
 - [CertMgrAddCertificateToStore, 38](#)
 - [CertMgrAddMetaDataToCert, 38](#)
 - [CertMgrCertField, 35](#)
 - [CertMgrCertLinkedList, 36](#)
 - [CertMgrCertStatus, 36](#)
 - [CertMgrCipherAlgo, 36](#)
 - [CertMgrDisableCertificate, 39](#)
 - [CertMgrEnableCertificate, 39](#)
 - [CertMgrErrorCodeToMessage, 40](#)
 - [CertMgrExtensionsFldData, 36](#)
 - [CertMgrExtractCertificateData, 41](#)
 - [CertMgrGenerateFilBuffSignature, 41](#)
 - [CertMgrGenerateMemBuffSignature, 42](#)
 - [CertMgrGetCertificateType, 43](#)
 - [CertMgrIssuerFldData, 36](#)
 - [CertMgrIssuerUIDFldData, 36](#)
 - [CertMgrPublicKeyFldData, 36](#)
 - [CertMgrPurgeMetaDataFromCert, 44](#)
 - [CertMgrReleaseCertificateData, 44](#)
 - [CertMgrRemoveCertificateFromStore, 45](#)
 - [CertMgrRemoveMetaDataFromCert, 45](#)
 - [CertMgrResult, 36](#)
 - [CertMgrRetrieveCertFromStore, 46](#)
 - [CertMgrRetrieveCertStatus, 46](#)
 - [CertMgrRetrieveMetaDataForCert, 47](#)
 - [CertMgrSerialNoFldData, 36](#)
 - [CertMgrSigAlgoFldData, 36](#)
 - [CertMgrSigType, 36](#)
 - [CertMgrStorageId, 37](#)
 - [CertMgrSubjectFldData, 37](#)
 - [CertMgrSubjectUIDFldData, 37](#)
 - [CertMgrVerifyCertificate, 48](#)
 - [CertMgrVerifyFBufSigWithCert, 48](#)
 - [CertMgrVerifyFileBufferSignature, 49](#)
 - [CertMgrVerifyMBufSigWithCert, 50](#)
 - [CertMgrVerifyMemBufferSignature, 51](#)
 - [CertMgrVersionFldData, 37](#)
 - [CERTPROCESS_START_ERRCODE, 33](#)
 - [CERTREVOC_STATUS_START_ -
 \[ERRCODE, 33\]\(#\)](#)
 - [CERTSIGNATURE_START_ERRCODE, 33](#)
 - [CERTSTORE_START_ERRCODE, 33](#)
 - [ERR_BROKEN_EXTERN_CHAIN, 33](#)
 - [ERR_BUFFER_TOO_SMALL, 34](#)
 - [ERR_DUPLICATE_CERTIFICATE, 34](#)
 - [ERR_FILE_IO_ERROR, 34](#)
 - [ERR_INVALID_CERT_ID, 34](#)

- ERR_INVALID_CERTIFICATE, [34](#)
- ERR_INVALID_SIGNATURE, [34](#)
- ERR_NO_MORE_CERTIFICATES, [34](#)
- ERR_NO_ROOT_CERTIFICATE, [34](#)
- ERR_STORE_IS_EMPTY, [34](#)
- ERR_STORE_IS_FULL, [34](#)
- ERR_STORE_IS_UNAVAILABLE, [35](#)
- ERR_UNKNOWN_RESULT_CODE, [35](#)
- ERR_UNSUPPORTED_HASH_TYPE, [35](#)
- ERR_UNSUPPORTED_KEY_TYPE, [35](#)
- ERR_UNSUPPORTED_OPERATION, [35](#)
- INT16, [37](#)
- INT32, [37](#)
- INT8, [37](#)
- MAX_ERRCODES_PER_COMPONENT, [35](#)
- MAX_RESERVED_ERR_CODE, [35](#)
- TRUE, [35](#)
- UINT16, [37](#)
- UINT32, [37](#)
- UINT8, [37](#)
- CERTMGR_CERTFLD_EXTENSIONS
 - CertMgr.h, [31](#)
- CERTMGR_CERTFLD_ISSUER
 - CertMgr.h, [31](#)
- CERTMGR_CERTFLD_ISSUER_UID
 - CertMgr.h, [31](#)
- CERTMGR_CERTFLD_PUBLIC_KEY
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SERIALNO
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SIG_ALGO
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SIGNATURE
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SIGNED_DATA
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SUBJECT
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_SUBJECT_UID
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_VALIDITY
 - CertMgr.h, [32](#)
- CERTMGR_CERTFLD_VERSION
 - CertMgr.h, [32](#)
- CERTMGR_CERTID_HASH_SIZE
 - CertMgr.h, [32](#)
- CERTMGR_CIPHER_RSA
 - CertMgr.h, [32](#)
- CERTMGR_CRYPTOGRAPHY
 - CertMgrPlugin.h, [69](#)
- CERTMGR_DEBUG
 - CertMgrDebug.h, [65](#)
- CERTMGR_DIGITALSIGNATURE
 - CertMgrPlugin.h, [69](#)
- CERTMGR_HASH_MD2_RSA
 - CertMgr.h, [33](#)
- CERTMGR_HASH_MD5_RSA
 - CertMgr.h, [33](#)
- CERTMGR_HASH_SHA1_RSA
 - CertMgr.h, [33](#)
- CERTMGR_INIT_CONTEXT
 - CertMgr.h, [33](#)
- CERTMGR_NOT_REVOKED
 - CertMgrRevocationStatus.h, [72](#)
- CERTMGR_PROCESSING
 - CertMgrPlugin.h, [69](#)
- CERTMGR_REVOKED
 - CertMgrRevocationStatus.h, [72](#)
- CERTMGR_START_ERRCODE
 - CertMgr.h, [33](#)
- CERTMGR_STATUS
 - CertMgrPlugin.h, [69](#)
- CERTMGR_STORE
 - CertMgrPlugin.h, [69](#)
- CertMgrAddCertificateToStore
 - CertMgr.h, [38](#)
- CertMgrAdditionalFields, [12](#)
- CertMgrAdditionalFields
 - fields, [12](#)
 - numberOfFields, [12](#)
- CertMgrAddMetaDataToCert
 - CertMgr.h, [38](#)
- CertMgrCertDescriptor, [13](#)
- CertMgrCertDescriptor
 - type, [13](#)
- CertMgrCertField
 - CertMgr.h, [35](#)
- CertMgrCertFieldDesc, [14](#)
- CertMgrCertFieldDesc
 - data, [14](#)
 - dataSize, [14](#)
 - name, [14](#)
- CertMgrCertId, [15](#)
- CertMgrCertId
 - certificateNo, [15](#)
 - storeId, [15](#)
 - type, [15](#)
- CertMgrCertIsMyCertificate
 - CertMgrCertProcessing.h, [53](#)
- CertMgrCertLinkedList
 - CertMgr.h, [36](#)
- CertMgrCertProcessing.h, [52](#)
- CertMgrCertProcessing.h
 - CertMgrCertIsMyCertificate, [53](#)
 - CertMgrCertProcessingCertMatch, [53](#)
 - CertMgrCertProcessingDecodeCert, [54](#)
 - CertMgrCertProcessingMatchField, [54](#)
 - CertMgrCertProcessingRelCrtDta, [55](#)

- ERR_CMPROC_INVALID_CERTIFICATE, 52
- ERR_CMPROC_UNKNOWN_CERTIFICATE, 52
- CertMgrCertProcessingCertMatch
 - CertMgrCertProcessing.h, 53
- CertMgrCertProcessingDecodeCert
 - CertMgrCertProcessing.h, 54
- CertMgrCertProcessingMatchField
 - CertMgrCertProcessing.h, 54
- CertMgrCertProcessingRelCrtDta
 - CertMgrCertProcessing.h, 55
- CertMgrCertStatus
 - CertMgr.h, 36
- CertMgrCipherAlgo
 - CertMgr.h, 36
- CertMgrCryptoDecrypt
 - CertMgrCryptography.h, 58
- CertMgrCryptoEncrypt
 - CertMgrCryptography.h, 59
- CertMgrCryptoFinaliseHash
 - CertMgrCryptography.h, 60
- CertMgrCryptoGenerateHash
 - CertMgrCryptography.h, 61
- CertMgrCryptoGetHashBuffSize
 - CertMgrCryptography.h, 61
- CertMgrCryptography.h, 57
- CertMgrCryptography.h
 - CertMgrCryptoDecrypt, 58
 - CertMgrCryptoEncrypt, 59
 - CertMgrCryptoFinaliseHash, 60
 - CertMgrCryptoGenerateHash, 61
 - CertMgrCryptoGetHashBuffSize, 61
 - CertMgrCryptoInitHash, 62
 - CertMgrCryptoProcessHash, 63
- ERR_CMCRYPT_ALGO_NOT_SUPPORTED, 58
- ERR_CMCRYPT_BUFF_TOO_SMALL, 58
- ERR_CMCRYPT_NO_MEMORY, 58
- ERR_CMCRYPT_UNKNOWN_HASH_TYPE, 58
- CertMgrCryptoInitHash
 - CertMgrCryptography.h, 62
- CertMgrCryptoProcessHash
 - CertMgrCryptography.h, 63
- CertMgrCtx, 16
- CertMgrCtx
 - firstCall, 16
 - lastRetrieved, 16
 - lastRetrievedId, 17
 - open, 16
 - retrieveByField, 16
 - retrieveByType, 16
 - searchTarget, 16
 - type, 16
- CertMgrDebug.h, 65
- CertMgrDebug.h
 - CERTMGR_DEBUG, 65
- CertMgrDisableCertificate
 - CertMgr.h, 39
- CertMgrEnableCertificate
 - CertMgr.h, 39
- CertMgrErrorCodeToMessage
 - CertMgr.h, 40
- CertMgrExit
 - CertMgrPlugin.h, 70
- CertMgrExtensionsFldData
 - CertMgr.h, 36
- CertMgrExtractCertificateData
 - CertMgr.h, 41
- CertMgrFilBuff, 18
- CertMgrFilBuff
 - fileName, 18
 - fileOffset, 18
 - size, 18
- CertMgrFindNextPlugin
 - CertMgrPlugin.h, 70
- CertMgrFindPlugin
 - CertMgrPlugin.h, 70
- CertMgrGenerateFilBuffSignature
 - CertMgr.h, 41
- CertMgrGenerateMemBuffSignature
 - CertMgr.h, 42
- CertMgrGetCertificateType
 - CertMgr.h, 43
- CertMgrInit
 - CertMgrPlugin.h, 70
- CertMgrInternal.h, 66
- CertMgrIssuerFldData
 - CertMgr.h, 36
- CertMgrIssuerUIDFldData
 - CertMgr.h, 36
- CertMgrKeyDescr, 19
- CertMgrKeyDescr
 - key, 19
- CertMgrMandatoryFields, 20
- CertMgrMemBuff, 21
- CertMgrMemBuff
 - data, 21
 - firstFree, 21
 - size, 21
- CertMgrMetaDataId, 22
- CertMgrMetaDataId
 - metaDataType, 22
 - ownerApplication, 22
- CertMgrPlugin.h, 69
- CERTMGR_CRYPTOGRAPHY, 69
- CERTMGR_DIGITALSIGNATURE, 69

- CERTMGR_PROCESSING, 69
- CERTMGR_STATUS, 69
- CERTMGR_STORE, 69
- CertMgrPlugin.h
 - CertMgrExit, 70
 - CertMgrFindNextPlugin, 70
 - CertMgrFindPlugin, 70
 - CertMgrInit, 70
 - CertMgrPlugInType, 69
- CertMgrPlugInDescr, 23
- CertMgrPlugInDescr
 - algorithm, 23
 - plugInLocation, 23
- CertMgrPlugInType
 - CertMgrPlugin.h, 69
- CertMgrPublicKeyFldData
 - CertMgr.h, 36
- CertMgrPurgeMetaDataFromCert
 - CertMgr.h, 44
- CertMgrReleaseCertificateData
 - CertMgr.h, 44
- CertMgrRemoveCertificateFromStore
 - CertMgr.h, 45
- CertMgrRemoveMetaDataFromCert
 - CertMgr.h, 45
- CertMgrResult
 - CertMgr.h, 36
- CertMgrRetrieveCertFromStore
 - CertMgr.h, 46
- CertMgrRetrieveCertStatus
 - CertMgr.h, 46
- CertMgrRetrieveMetaDataForCert
 - CertMgr.h, 47
- CertMgrRevocationStatus
 - CertMgrRevocationStatus.h, 71
- CertMgrRevocationStatus.h, 71
 - CERTMGR_NOT_REVOKED, 72
 - CERTMGR_REVOKED, 72
- CertMgrRevocationStatus.h
 - CertMgrRevocationStatus, 71
 - CertMgrRevocationStatusVerify, 72
 - ERR_CMREVOC_INVALID_CERTIFICATE, 71
 - ERR_CMREVOC_NO_REVOC_STATUS, 71
- CertMgrRevocationStatusVerify
 - CertMgrRevocationStatus.h, 72
- CertMgrSerialNoFldData
 - CertMgr.h, 36
- CertMgrSigAlgoFldData
 - CertMgr.h, 36
- CertMgrSignature.h, 73
- CertMgrSignature.h
 - CertMgrSignatureGenerate, 74
 - CertMgrSignatureInitContext, 75
 - CertMgrSignatureInitialize, 76
 - CertMgrSignatureLoadCryptoInfo, 77
 - CertMgrSignatureReleaseContext, 77
 - CertMgrSignatureVerify, 78
 - ERR_CMSIG_NO_MEMORY, 74
 - ERR_CMSIG_UNSUPPORTED_HASH_TYP, 74
- CertMgrSignatureGenerate
 - CertMgrSignature.h, 74
- CertMgrSignatureInitContext
 - CertMgrSignature.h, 75
- CertMgrSignatureInitialize
 - CertMgrSignature.h, 76
- CertMgrSignatureLoadCryptoInfo
 - CertMgrSignature.h, 77
- CertMgrSignatureReleaseContext
 - CertMgrSignature.h, 77
- CertMgrSignatureVerify
 - CertMgrSignature.h, 78
- CertMgrSigType
 - CertMgr.h, 36
- CertMgrStorageId
 - CertMgr.h, 37
- CertMgrStore.h, 80
- CertMgrStore.h
 - CertMgrStoreDisableCertificate, 81
 - CertMgrStoreEnableCertificate, 81
 - CertMgrStoreInitialize, 81
 - CertMgrStoreRemoveAllUserCertificates, 82
 - CertMgrStoreRemoveCertificate, 82
 - CertMgrStoreRetrieveCertificate, 83
 - CertMgrStoreRetrieveCertificateStatus, 84
 - CertMgrStoreStoreCertificate, 84
 - ERR_CMSTORE_IS_FULL, 80
- CertMgrStoreDisableCertificate
 - CertMgrStore.h, 81
- CertMgrStoreEnableCertificate
 - CertMgrStore.h, 81
- CertMgrStoreInitialize
 - CertMgrStore.h, 81
- CertMgrStoreRemoveAllUserCertificates
 - CertMgrStore.h, 82
- CertMgrStoreRemoveCertificate
 - CertMgrStore.h, 82
- CertMgrStoreRetrieveCertificate
 - CertMgrStore.h, 83
- CertMgrStoreRetrieveCertificateStatus
 - CertMgrStore.h, 84
- CertMgrStoreStoreCertificate
 - CertMgrStore.h, 84
- CertMgrSubjectFldData
 - CertMgr.h, 37
- CertMgrSubjectUIDFldData
 - CertMgr.h, 37

- CertMgrValidityFldData, [24](#)
- CertMgrVerifyCertificate
 - CertMgr.h, [48](#)
- CertMgrVerifyFBufSigWithCert
 - CertMgr.h, [48](#)
- CertMgrVerifyFileBufferSignature
 - CertMgr.h, [49](#)
- CertMgrVerifyMBufSigWithCert
 - CertMgr.h, [50](#)
- CertMgrVerifyMemBufferSignature
 - CertMgr.h, [51](#)
- CertMgrVersionFldData
 - CertMgr.h, [37](#)
- CERTPROCESS_START_ERRCODE
 - CertMgr.h, [33](#)
- CERTREVOC_STATUS_START_ERRCODE
 - CertMgr.h, [33](#)
- CERTSIGNATURE_START_ERRCODE
 - CertMgr.h, [33](#)
- CERTSTORE_START_ERRCODE
 - CertMgr.h, [33](#)
- data
 - CertMgrCertFieldDesc, [14](#)
 - CertMgrMemBuff, [21](#)
- dataSize
 - CertMgrCertFieldDesc, [14](#)
- ERR_BROKEN_EXTERN_CHAIN
 - CertMgr.h, [33](#)
- ERR_BUFFER_TOO_SMALL
 - CertMgr.h, [34](#)
- ERR_CMCRYPT_ALGO_NOT_SUPPORTED
 - CertMgrCryptography.h, [58](#)
- ERR_CMCRYPT_BUFF_TOO_SMALL
 - CertMgrCryptography.h, [58](#)
- ERR_CMCRYPT_NO_MEMORY
 - CertMgrCryptography.h, [58](#)
- ERR_CMCRYPT_UNKNOWN_HASH_TYPE
 - CertMgrCryptography.h, [58](#)
- ERR_CMPROC_INVALID_CERTIFICATE
 - CertMgrCertProcessing.h, [52](#)
- ERR_CMPROC_UNKNOWN_CERTIFICATE
 - CertMgrCertProcessing.h, [52](#)
- ERR_CMREVOC_INVAL_CERTIFICATE
 - CertMgrRevocationStatus.h, [71](#)
- ERR_CMREVOC_NO_REVOC_STATUS
 - CertMgrRevocationStatus.h, [71](#)
- ERR_CMSIG_NO_MEMORY
 - CertMgrSignature.h, [74](#)
- ERR_CMSIG_UNSUPPORTED_HASH_TYP
 - CertMgrSignature.h, [74](#)
- ERR_CMSTORE_IS_FULL
 - CertMgrStore.h, [80](#)
- ERR_DUPLICATE_CERTIFICATE
 - CertMgr.h, [34](#)
- ERR_FILE_IO_ERROR
 - CertMgr.h, [34](#)
- ERR_INVALID_CERT_ID
 - CertMgr.h, [34](#)
- ERR_INVALID_CERTIFICATE
 - CertMgr.h, [34](#)
- ERR_INVALID_SIGNATURE
 - CertMgr.h, [34](#)
- ERR_NO_MORE_CERTIFICATES
 - CertMgr.h, [34](#)
- ERR_NO_ROOT_CERTIFICATE
 - CertMgr.h, [34](#)
- ERR_STORE_IS_EMPTY
 - CertMgr.h, [34](#)
- ERR_STORE_IS_FULL
 - CertMgr.h, [34](#)
- ERR_STORE_IS_UNAVAILABLE
 - CertMgr.h, [35](#)
- ERR_UNKNOWN_RESULT_CODE
 - CertMgr.h, [35](#)
- ERR_UNSUPPORTED_HASH_TYPE
 - CertMgr.h, [35](#)
- ERR_UNSUPPORTED_KEY_TYPE
 - CertMgr.h, [35](#)
- ERR_UNSUPPORTED_OPERATION
 - CertMgr.h, [35](#)
- fields
 - CertMgrAdditionalFields, [12](#)
- fileName
 - CertMgrFilBuff, [18](#)
- fileOffset
 - CertMgrFilBuff, [18](#)
- firstCall
 - CertMgrCtx, [16](#)
- firstFree
 - CertMgrMemBuff, [21](#)
- INT16
 - CertMgr.h, [37](#)
- INT32
 - CertMgr.h, [37](#)
- INT8
 - CertMgr.h, [37](#)
- key
 - CertMgrKeyDescr, [19](#)
- lastRetrieved
 - CertMgrCtx, [16](#)
- lastRetrievedId
 - CertMgrCtx, [17](#)

MAX_ERRCODES_PER_COMPONENT

CertMgr.h, [35](#)

MAX_RESERVED_ERR_CODE

CertMgr.h, [35](#)

metaDataType

CertMgrMetaDataId, [22](#)

name

CertMgrCertFieldDesc, [14](#)

next

_CertMgrCertLinkedList, [11](#)

numberOfFields

CertMgrAdditionalFields, [12](#)

open

CertMgrCtx, [16](#)

ownerApplication

CertMgrMetaDataId, [22](#)

plugInLocation

CertMgrPlugInDescr, [23](#)

retrieveByField

CertMgrCtx, [16](#)

retrieveByType

CertMgrCtx, [16](#)

searchTarget

CertMgrCtx, [16](#)

size

CertMgrFilBuff, [18](#)

CertMgrMemBuff, [21](#)

storeId

CertMgrCertId, [15](#)

TRUE

CertMgr.h, [35](#)

type

CertMgrCertDescriptor, [13](#)

CertMgrCertId, [15](#)

CertMgrCtx, [16](#)

UINT16

CertMgr.h, [37](#)

UINT32

CertMgr.h, [37](#)

UINT8

CertMgr.h, [37](#)